uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática

2017 - 2018

Trabajo Fin de Grado

"Extensión funcional de un SGBD NoSQL por aplicación de técnicas difusas"

Ainara Del Barco Ventura

Tutor/es

Francisco Javier Calle Gómez

Leganés, 2018

RESUMEN

En este proyecto realizamos un trabajo de investigación sobre la posibilidad de aumentar las operaciones que ofrece un gestor de base de datos NoSQL a través del uso de técnicas difusas. Para ello realizamos un estudio de todas las técnicas difusas que se han ideado hasta el momento y en qué ámbitos relacionados con el almacenamiento de la información se han aplicado para después aplicarlos a nuestro caso.

Se desarrolla por tanto un sistema que aplica las técnicas difusas estudiadas a las operaciones de inserción, borrado y recuperación de una base de datos NoSQL orientada a documentos, concretamente MongoDB.

Se espera que este trabajo sirva como predecesor para otros en un futuro, siendo el primer granito de arena para desarrollar un amplio conjunto de herramientas que apliquen técnicas difusas a las bases de datos NoSQL y que de esta manera se puedan encontrar aplicaciones al estudio y análisis de información, que principalmente fue el objetivo con el que esas bases de datos fueron creadas.

Palabras clave: Técnicas difusas, NoSQL, MongoDB, Operaciones difusas, aproximación de cadenas, pertenencia difusa, ampliación de operaciones.

DEDICATORIA

Primero de todo, me gustaría comenzar esta dedicatoria con un agradecimiento a mis padres, que han depositado en mi toda su confianza y me han dado todo su apoyo desde el momento en el que decidí que gran parte de mi futuro me lo iba a pasar estudiando. Por enseñarme a seguir mis metas y que todo aquello en lo que depositamos esfuerzo acaba dando frutos.

Agradecer también a mi hermano, que siempre tiene una broma que contar y una sonrisa que sacarte. Además de su preocupación constante por saber cuando iba a finalizar este trabajo y sus visitas periódicas que me animaban a continuar.

A mi tutor, Francisco Javier Calle, que además de despertarme la curiosidad por la teoría relacionada con el almacenamiento de la información (de aquí surge este trabajo) cuando fue mi profesor, también me ha guiado y ayudado a completar este trabajo.

A mis amigos de toda la vida, que me han apoyado en los mejores y peores momentos durante la carrera y durante mi día a día. Siempre tenían palabras de ánimo cuando estaba estresada por entregas finales o exámenes. Preparaos porque vais a tener que seguir animándome mientras dure el máster que voy a cursar en los próximos años. Es un placer teneros en mi vida: María, Mario y Patricia.

A todas las amistades que he hecho durante mi paso por la universidad: Fermín, Rubén, Chris, Javi, Dani, Quique... sin vosotros las tardes de estudio y prácticas en la universidad de 9 a 9, habrían sido mucho más difíciles de superar. Juntos sobrevivimos al primer año de carrera en la 7.0.J06 y estoy segura que sobreviviremos a muchas más aventuras.

Y por supuesto, a la persona que comenzó formando parte del grupo de amigos que hice en la universidad y que ha acabado siendo algo más que un amigo. Miguel, sin ti estoy segura de que no habría podido con todo lo que he vivido a lo largo de estos años. Gracias por tu apoyo, que me ha resultado indispensable, el cariño y la paciencia infinita que tienes conmigo, porque se que a veces es complicado aguantarme. Ahora nos toca seguir con una nueva etapa y espero que la quieras seguir compartiendo conmigo. Solo puedo decirte: Gracias por todo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En la actualidad vivimos en una sociedad interconectada en la que por unos medios u otros la cantidad de información de la que disponemos no hace más que crecer. El auge de las redes sociales, foros, aplicaciones móviles, web, etc han provocado que hoy en día dispongamos de una gran cantidad de datos de distintas naturalezas.

En cifras proporcionadas por las empresas "We Are Social" y "Hootsuite" de un estudio que se realizó en Enero de este año (2018), la cantidad de dispositivos conectados a en internet es aproximadamente de 7.593 billones [1] mientras que la población mundial en Enero de 2018 era de 7.576 billones. ¿Cómo puede ser que existan más dispositivos conectados que el número de personas existentes en el mundo? La respuesta es sencilla; al encontrarnos en la era de la informatización ha producido este fenómeno, la cantidad de dispositivos conectados a la red supera (y esta cifra, previsiblemente, continuará aumentando a gran velocidad) a la población total mundial. Además todos estos dispositivos cuentan con una conexión a internet, lo que produce que generen infinidad de datos que se deben almacenar, pero ¿cómo almacenarlos?.

Por un lado se pueden utilizar las bases de datos tradicionales orientadas a propósitos específicos que, por su naturaleza, los datos que se utilizan presentan unas dimensiones relativamente manejable. Pero esto no se produce para todos los casos, ya que cada vez es mayor el uso de las bases de datos orientadas a procesamiento analítico, en los que el volumen de datos es superior a lo que las tecnologías tradicionales pueden soportar. Además, se incorpora el requisito de minimizar el tiempo de respuesta (a menudo con el objetivo de procesar en "tiempo real") lo que complica la búsqueda de soluciones.

La naturaleza de este tipo de procesamiento analítico relaja en cierta medida estos requisitos, ya que, tal y como dicta la "ley de los grandes números" [2], podemos permitirnos prescindir de una cierta parte de los datos sin que esto afecte al resultado, al menos, de un modo estadísticamente significativo. Esto puede ser aprovechado para aplicar un enfoque distribuido en el que la relajación del parámetro de la exhaustividad puede compensar las exigencias del teorema CAP: la imposibilidad de alcanzar un sistema distribuido manteniendo al mismo tiempo la disponibilidad, la consistencia y la tolerancia a fallos. Se plantea así la distribución como solución plausible al almacenamiento masivo requerido, y el procesamiento paralelo como solución eficiente para el análisis de tal tamaño de datos. Por tanto la tecnología que ha llenado este espacio se ha denominado NoSQL (del inglés Not Only SQL) y se nombrarán de esta forma a partir de ahora a lo largo del documento.

La pregunta que surge en este momento es por qué y para qué analizar todos esos datos. La respuesta es simple: si se dispone de la capacidad de analizar los datos producidos por todos los dispositivos conectados a internet entonces se podrá extraer información valiosa de dichos datos y aplicarla a diversos entornos; el entorno financiero, la capacidad

de personalización (ofrecer a cada usuario información de interés para él) e incluso el marketing. Casi cualquier área a la que se pueda aplicar este análisis será susceptible de mejora gracias a un estudio de los datos que la afecten directamente.

No se entra en detalles de cómo se realizan los análisis de tantísimos datos ya que excede lo que se pretende exponer en este punto. Basta con decir que son las herramientas que proporcionan las bases de datos NoSQL, las que permiten que este análisis sea posible (en conjunto por supuesto, de profesionales que saben hacer uso de ella). Pero normalmente, los datos que se pretenden analizar no contienen información precisa, ya que muchas veces se trata de información vaga, o en otras palabras, que para el conocimiento humano resulta fácilmente comprensible pero no para un ordenador. Es por esto que toman importancia otras tecnologías que son capaces de analizar este tipo de información imprecisa y reciben el nombre de técnicas difusas.

Aunque parezca sorprendente, no existen (como veremos más adelante) muchos modelos tecnológicos que contemplen el uso generalizado de estas técnicas para tratar la información. Como se ha comentado, comúnmente se utilizan las operaciones y herramientas que proveen los gestores de base de datos de manera nativa y se deja de lado la opción de analizar los datos con técnicas difusas con las que quizá, se dispondría de ciertas ventajas a la hora de realizar un análisis más profundos de los datos. Es por eso que este trabajo tiene como motivación unir ambas facetas, las bases de datos NoSQL y las técnicas difusas para así cubrir esta situación y ver si es posible utilizar dichas técnicas para expandir las opciones que ofrece un gestor NoSQL actual.

Dicho esto se marca un único objetivo en este proyecto; la adición de nuevas operaciones a un gestor de base de datos NoSQL mediante el uso de estas técnicas como primera aproximación al uso de las técnicas difusas en este ámbito. Este objetivo dista mucho de generar un producto final que se pueda aplicar directamente a las bases de datos NoSQL y provea de multitud de herramientas difusas, ya que realizar un estudio completo de la tecnología aplicándola de forma satisfactoria al análisis de gran cantidad de datos excede en sobremanera la finalidad del proyecto fin de carrera, que es la razón por la que realizamos este trabajo. Por ello simplemente se pretende estudiar una primera base que sirva de predecesora para posibles trabajos futuros que se apliquen en este ámbito.

El presente documento consta de ocho grandes capítulos los cuales se mencionan a continuación a modo de resumen:

- En el capítulo 1 : "*Introducción*", se realiza un primer acercamiento al tema que va a tratar nuestro trabajo, exponiendo brevemente la motivación y los objetivos del mismo.

- En el capítulo 2 : "*Estado del arte*", se presenta en primer lugar la teoría relacionada con las técnicas difusas como antecedentes de la investigación y el efecto que han tenido dichas técnicas en el marco de las bases de datos. Después se presenta el marco actual en el que se introduce el concepto de una base de datos NoSQL y las

aportaciones de otros investigadores sobre la aplicación de las técnicas difusas a este tipo de base de datos. Además se aporta el marco regulador del proyecto y el marco socieconómico en el que tiene cabida.

- En el capítulo 3 : "*Definición del proyecto*", se realiza un análisis del problema teniendo en cuenta la exposición realizada en el capítulo anterior, sobre el estado en el que se encuentra la cuestión que nos ocupa. Además se realiza un análisis de la tecnología sobre la que podemos aplicar este trabajo y se selecciona aquella en la que finalmente se va a desarrollar atendiendo a diversos criterios. Finalmente se realizan las tareas de análisis propiamente dichas; establecimiento de requisitos, de la planificación y del plan de pruebas.

- En el capítulo 4 : "*Ejecución del proyecto*" se lleva a la práctica la teoría expuesta anteriormente y se expone la implementación que se ha realizado haciendo uso de la tecnología pertinente y explicada en el anterior capítulo.

- En el capítulo 5 : "*Descripción de resultados*" se llevan a cabo pruebas para evaluar el funcionamiento del trabajo desarrollado. Además se presenta la ejecución real del proyecto comparándolo con la planificación y costes propuestos en el capítulo 3 y analizando las posibles desviaciones.

- En el capítulo 6 : "*Conclusiones*" se detallan las conclusiones obtenidas fruto de la realización de este proyecto así como los trabajos futuros que pueden derivar del mismo.

- En el capítulo 7 : "*English Summary*" se realiza un resumen completo del trabajo escrito en inglés, aunque se recomienda que para la comprensión completa del trabajo se lea la parte en castellano.

En último lugar se encuentra la bibliografía, seguida del glosario de términos y los anexos.

2. ESTADO DEL ARTE

En este capítulo se presentan los antecedentes y los fundamentos teóricos sobre los que se basará nuestra propuesta. En primer lugar estudiaremos los factores previos que inciden directamente en el objetivo y desarrollo de este trabajo en lo que a técnicas difusas se refiere, mencionando principalmente dos de ellas: La técnica de conjuntos difusos y la recuperación de información gracias a la aproximación de cadenas, que componen la teoría de partida en la que englobamos nuestra propuesta. Estudiaremos también el marco actual relacionado con la aplicación de estas técnicas difusas a las bases de datos, y más concretamente a los sistemas NoSQL que son los principales afectados por la propuesta que nos ocupa. Finalmente se aportan las conclusiones extraídas del estado del arte y el marco regulador y socio-económico que afecta al trabajo que pretendemos desarrollar.

2.1. Antecedentes de la investigación.

Existen gran variedad de técnicas difusas, que hacen referencia a todas las estrategias de representación del conocimiento y/o análisis de la información que tienen como punto de partida el modelo que propuso el matemático Lotfi A. Zadeh. Fue el primero en formular el concepto de conjunto difuso [3] tras percatarse de lo que él llamó principio de incompatibilidad: "Conforme la complejidad de un sistema aumenta, nuestra capacidad para ser precisos y construir instrucciones sobre su comportamiento disminuye hasta el umbral más allá del cual, la precisión y el significado son características excluyentes".

2.1.1. Teoría de los conjuntos difusos.

Para A. Zadeh, la lógica clásica o bivaluada no permitía representar y resolver todos los problemas del mundo real, puesto que existe mucho conocimiento no-perfecto, es decir, conocimiento vago, inexacto, impreciso o probabilístico por naturaleza. Fue por esto que tras la formulación de la teoría de conjuntos difusos, se popularizó el uso de esta técnica cuando la complejidad del problema a resolver era muy alta y no existían modelos matemáticos precisos, o cuando se usaban definiciones y conocimiento impreciso o subjetivo. Gracias a estos métodos, surgió una forma de representar la imprecisión y la incertidumbre que hasta ese momento no se había planteado y no era posible realizar con la lógica clásica.

Supongamos por ejemplo que nos encontramos ante un conjunto de personas a las que intentamos agrupar en función de su altura, clasificándolas en *altas* o *bajas*. Pero, ¿Qué es una persona *baja*?
Con la lógica bivaluada nos vemos en la obligación de establecer un umbral para el cual se decide si una persona pertenece a un conjunto u otro, por ejemplo, para todas aquellas

personas que midan más de 1.75m diremos que se trata de una persona *alta* mientras que para las que no superen dicha marca pertenecerán al conjunto de personas *bajas*. Esto da lugar a un resultado demasiado restrictivo: que consideremos de la misma manera a una persona que mide 1.74m que a otra que mide 1.45m, ya que ambas están contenidas dentro del conjunto de personas *bajas*. No obstante, la realidad es que estas transiciones son mucho más suaves puesto que muchos conceptos que manejamos los humanos a menudo no tienen una definición clara. El concepto para calificar a alguien como perteneciente al conjunto de persona *baja* es un claro ejemplo de ello, ya que se realiza de forma gradual según nos vamos alejando del valor umbral, es decir, una persona que mide 1.70m no pertenece en el mismo grado al conjunto de personas *bajas* que la persona que mide 1.40m. En rasgos generales, es en esta premisa en la que se apoya la teoría de conjuntos difusos.

2.1.1.1. Formulación de la teoría de los conjuntos difusos

De manera más formal podemos explicar los rasgos principales en los que se apoya esta teoría. Se define la pertenencia de un elemento x a un conjunto A haciendo uso de las llamadas funciones de pertenencia $\mu_A(x)$. Con la lógica clásica, el elemento x puede pertenecer al conjunto A dando como resultado que el valor de $\mu_A(x)$ sea igual a 1 o no pertenecer al conjunto dando lugar a que el valor de la función de pertenencia sea igual a 0. Expresado de forma matemática esto es:

$$\mu_A(x) = \begin{cases} 0 & si \quad x \notin A \\ 1 & si \quad x \in A \end{cases}$$

Sin embargo con la lógica difusa, se relaja la asignación de pertenencia o no pertenencia que resulta muy restrictiva y se permite que el elemento x pertenezca al conjunto A tomando $\mu_A(x)$ valores en el intervalo $[0, 1]$ que indican el grado de pertenencia del elemento en cuestión al conjunto. De esta manera cuanto más cercanos estén los valores de $\mu_A(x)$ a 1 mayor será la pertenencia del elemento x al conjunto A y viceversa. Expresado de forma matemática esto es:

$$\mu_A(x) \rightarrow [0, 1]$$

Por tanto, la principal diferencia que existe entre la lógica bivaluada y la difusa es que los valores que puede tomar la función de pertenencia para un conjunto clásico es 0 ó 1 mientras que los valores de la función de pertenencia de un conjunto difuso recae en todo el intervalo real $[0, 1]$. De esta manera un conjunto clásico puede definirse de varias formas (dependiendo de la condición de pertenencia al conjunto que se establezca) mientras que un conjunto difuso viene definido siempre por una función de pertenencia difusa.

2.1.1.2. Funciones de pertenencia difusa

Estas funciones pueden construirse libremente pero en la práctica siempre se usan las mismas para definir distintos tipos de problemas y representar la pertenencia de los datos de entrada a los conjuntos difusos. Algunos ejemplos de las funciones de pertenencia que más se usan son los siguientes:

- Función 'GAMMA' y 'L': Este tipo de funciones se utilizan para representar los valores extremos. Por ejemplo en el caso del conjunto difuso para determinar si un persona pertenece o no al conjunto *alto* tenemos la siguiente gráfica en la que a partir de la altura 1.50m comienza a incrementarse el grado de pertenencia hasta el valor 1.75m en el que el grado de pertenencia se sitúa en 1.

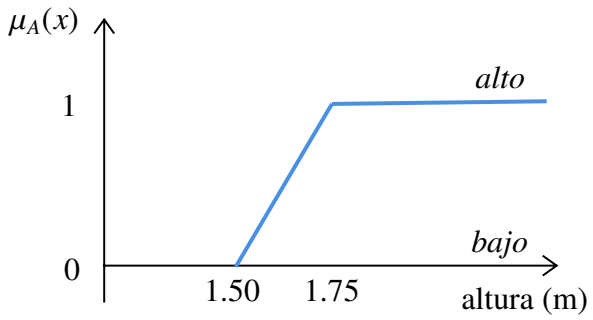

Fig. 2.1. Función de pertenencia 'GAMMA'

En el caso de la función 'L' simplemente se trata del inverso de la función anterior. Representa también los valores extremos, pero en este caso, el grado de pertenencia toma valor 1 cuando la persona pertenece al conjunto *bajo* y va disminuyendo conforme nos alejamos de la altura de 1.50 hasta llegar a 0 en el valor de altura 1.75m.

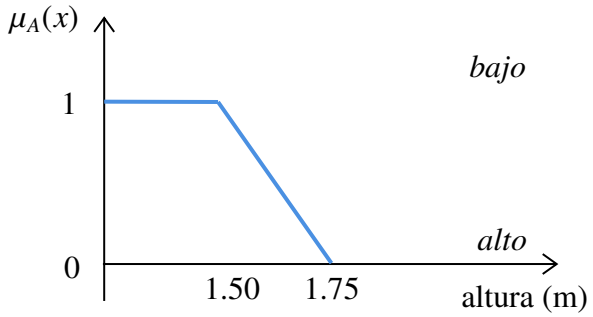

Fig. 2.2. Función de pertenencia 'L'

- Función 'LAMBDA' o triangular y 'PI' o trapezoidal: Este tipo de funciones se utilizan para representar valores intermedios. Como por ejemplo, podría ser el conjunto *mediana estatura* situado entre 1.50m y 1.70m. En el caso de la función

'LAMBDA' una persona pertenece al conjunto de *mediana estatura* con un grado de pertenencia 1 cuando alcanza el valor 1.60m, y dicho grado disminuye tanto para valores de altura inferiores y superiores como se aprecia en la gráfica.

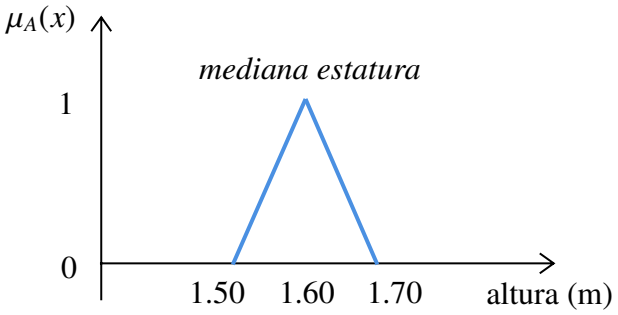

Fig. 2.3. Función de pertenencia 'LAMBDA' o triangular

En el caso de la función 'PI' o trapezoidal, la diferencia radica en que deja un margen de tolerancia alrededor del valor que se le asigna al conjunto de *mediana estatura* en este caso, de manera que para medidas superiores e inferiores a 1.60m el grado de pertenencia continúa siendo 1.

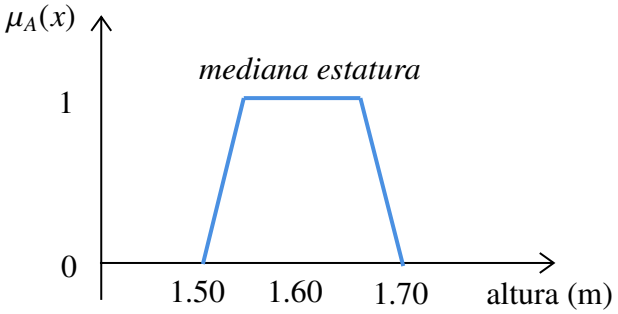

Fig. 2.4. Función de pertenencia 'PI' o trapezoidal

- También se suelen usar funciones más suaves que no sean lineales a trozos como las que se han expuesto anteriormente. Por ejemplo la función sigmoidal, la gaussiana, la pseudo-exponencial, etc.

Para finalizar la explicación de la teoría que engloba a los conjuntos difusos, destacamos que esta es mucho más extensa que lo expuesto en este punto, pero no se ha ahondado más en la teoría de la misma puesto que la pretensión de explicarla era dar unas pinceladas básicas sobre la teoría propuesta por Lofti A. Zadeh. En las referencias [4] y [5] se puede consultar más información a cerca de la teoría de conjuntos difusos.

2.1.1.3. Aplicación en sistemas gestores de datos

Gracias a esta teoría, se pudo representar el conocimiento que no era preciso y resolver problemas que hasta entonces era imposible plantear con la lógica clásica dando lugar a multitud de aplicaciones. Una de estas aplicaciones, que ha sido la principal impulsora de la propuesta que se describe en este trabajo fue el uso de la lógica difusa en las bases de datos relacionales.

Este tipo de bases de datos que implementan la lógica difusa propuesta por Zadeh, nacen para permitir dos objetivos principalmente. El primero, disponer de un sistema que sea capaz de almacenar información con incertidumbre a la par que información no difusa, y el segundo, poder consultar esta información de manera difusa o flexible. Surgen porque las bases de datos relacionales hasta ese momento, al igual que sucedía con la lógica clásica, resultan muy restrictivas y limitadas: no es posible realizar consultas con datos imprecisos, mientras que el lenguaje natural con el que trabajamos contiene una gran cantidad de este conocimiento. Véase por ejemplo, si queremos consultar el número de empleados cuyo salario es alto. Este tipo de consultas nos es imposible realizarla en un SGBD relacional al uso, ni tampoco almacenar la información difusa de que es un *salario alto*.

Siguiendo con esta premisa de introducir la lógica difusa en una base de datos relacional, surgieron distintos modelos como por ejemplo GEFRED. Este modelo fue introducido en el año 1994 por Medina J.M., Pons O. y Vila A, de la Universidad de Granada, en la revista *Information Sciences 76* [6], *87-109*.

Dichos modelos permitían realizar consultas de tipo FSQL (Fuzzy Structured Query Language) gracias a la introducción en el SGBD de una serie de herramientas entre las que se encuentran las siguientes:

- Etiquetas lingüísticas: Se introducen etiquetas que definen conocimiento difuso como por ejemplo pueden ser *alto*, *joven*...
- Expresiones condicionales: Permiten realizar comparaciones de forma difusa entre atributos, ya sean difusos o no. Son del tipo *mucho mayor que*, *aproximadamente igual*...
- Umbral de cumplimiento: Se puede establecer un umbral en el intervalo [0, 1] para el que se cumple la consulta difusa. Por ejemplo, seleccionar las personas *altas* con un umbral $\geq$ 0,6, selecciona todas las personas que se encuentren en dicho conjunto difuso con un grado de pertenencia mayor a 0,6.

Adicionalmente los SGBD que implementaban lógica difusa incorporaban toda la estructura necesaria para que tanto el almacenamiento de la información como la realización de las consultas difusas fueran posibles. Se puede consultar más información en[7] y [8] a cerca de todas las características que componen las bases de datos relacionales difusas.

Para finalizar, mencionar que este tipo de bases de datos permiten la recuperación de datos con tan solo una vaga descripción de aquello que queremos obtener, siendo esto un importante avance al unir el lenguaje natural y la organización de los datos como se conocía hasta ese momento. No obstante el uso de unos comparadores tan abstractos que hacía difícil la decisión sobre cual era el más adecuado, o la gran cantidad de parámetros que había que utilizar a la hora de realizar una consulta desembocaron en que estos sistemas no gozaran de mucha popularidad y no se generalizase su uso a pesar de su utilidad. Además la falta de especialistas en dicho campo, que resultaba novedoso, provocó también una falta de generalización a la hora de construir estos sistemas.

2.1.2. Aproximación de cadenas.

Otra técnica que tiene algunos puntos en común con la lógica difusa que propuso Zadeh es la llamada aproximación de cadenas (en inglés, approximate string matching).

En el caso de la lógica difusa, se busca una manera de representar el conocimiento imperfecto para poder trabajar con él. Este conocimiento, como se ha explicado en el punto anterior, se representa en forma de conjuntos y la lógica difusa permite categorizar un elemento y conocer el grado de pertenencia a dichos conjuntos mediante un valor dado en el intervalo $[0, 1]$. Cuando hablamos de aproximación de cadenas, nos encontramos ante una técnica que ofrece dos nociones diferentes: la resolución de un problema de equivalencia aplicado a una cadena de caracteres, o bien el problema de encontrar el grado de similitud que comparten dos cadenas.

2.1.2.1. Problema de similitud

Cuando hablamos de la técnica de aproximación de cadenas como la determinación de la similitud entre dos o más cadenas, este es el método que más se aproxima a las bases expuestas en el punto anterior, ya que plantea comparar dos cadenas de caracteres A y B y establecer cual es el grado de similitud entre ellas. En otras palabras, se busca establecer la pertenencia de la cadena A al conjunto difuso que forman las propiedades de la cadena B para determinar si ambas cadenas se pueden tratar como la misma cadena en un cierto grado comprendido en el intervalo $[0, 1]$. Este grado de pertenencia se determina gracias a la aplicación de distintos algoritmos sobre las cadenas que se pretenden comparar.

Esta técnica adquiere mucha importancia cuando hablamos de tareas como búsqueda de texto, clasificación, procesamiento estadístico del texto... Por ejemplo, ¿Qué ocurre si necesitamos buscar información sobre una persona en un sistema mediante la introducción de su nombre pero no conocemos como se encuentra este almacenado exactamente? Si nos equivocamos e introducimos el nombre de forma errónea el resultado no será el que esperamos, o incluso no obtendremos ningún resultado. Es en estos casos donde entra en juego la técnica de aproximación de cadenas, siendo bastante utilizada en los sistemas de recuperación de información como puede ser una base de datos. Al aplicarla, no necesi-

tamos conocer aquello que buscamos de forma precisa, por ejemplo, si podemos escribir el nombre de la persona, aunque sea de manera inexacta, el sistema recuperará la información que estamos buscando puesto que no necesita de una coincidencia idéntica para proporcionar el resultado de la consulta que hemos realizado.

Es principalmente debido a la aplicación de la resolución de este problema para la recuperación de información (que podemos calificar como recuperar la información de forma difusa), que distinguimos en un principio entre dos tipos de problemas de aproximación de cadenas: *offline* y *online* en función de la situación en la que estemos analizando la cadena. La primera se refiere a la situación en la que se busca una cadena en un sistema de información pero previamente se realiza un preprocesamiento del texto para poder generar un índice de búsqueda, mientras que en la segunda no existe este preprocesado del texto, sino que se permite preprocesar el patrón de búsqueda. En otras palabras, las técnicas de búsqueda *online* permiten realizar la búsqueda sin un índice, lo que las hace muy lentas cuando nos encontramos ante un sistema que contiene una gran cantidad de datos. Es por esto por lo que han surgido diversas técnicas que llevan a cabo una indexación del texto, lo que permite el almacenamiento de la información en un diccionario para que de esta forma sea más sencillo y preciso la recuperación de texto de forma difusa con técnicas *online* cuando se trata de un sistema que contiene grandes cantidades de datos y de esta manera no emplear grandes cantidades de tiempo en realizar primeramente la búsqueda en el sistema y sobre ella aplicar los algoritmos de aproximación de cadenas.

Dicho esto, los algoritmos que se usan en las técnicas de aproximación de cadenas para la recuperación difusa de la información, independientemente si se realiza *offline* u *online* pueden dividirse en dos tipos de algoritmos:

- **Basados en la distancia de edición**: Este tipo de algoritmos, cuentan de forma ponderada el número de modificaciones que hay que realizar en una cadena para que se transforme en otra. Algunos ejemplos de estos algoritmos son los que se muestran en la siguiente tabla:

Algoritmo	Descripción	Ejemplo
Levenshtein	El número mínimo de operaciones que deben realizarse sobre la cadena de texto para transformarla en otra. Estas operaciones pueden ser la eliminación, sustitución o inserción de un caracter.[9]	Inserción : cat → ca**r**t Eliminación: ca**r**t → cat Sustitución: **k**itten → **b**itten Ejemplo: book → back La distancia Levenshtein es 2.

Damerau-Levenshtein	Su funcionamiento es el mismo que el del algoritmo *Levenshtein* con la modificación de que se añade una nueva operación válida para el cálculo de la distancia: la transposición de los caracteres.	Transposición : c**at** → c**ta**
Longest Common Subsequence (LCS)	Encontrar la secuencia de caracteres más larga que comparten dos (o más) cadenas. No hay que confundirlo con el *Longest common substring* que a diferencia de lo que ocurre con la *LCS* las secuencias de caracteres deben ser consecutivas	Para las cadenas **AGCAT** y **AGCT** el valor de la distancia de la *LCS* es 1.
Jaro-Winkler	Mide el número mínimo de transposiciones que deben realizarse entre los caracteres de una cadena para transformarla en otra.	Es muy utilizada para detectar errores de escritura, por ejemplo My string ≈ My tsring

Tabla 2.1. Algoritmos basados en la distancia de edición

- **Basados en la distancia de *q-grams***: Este tipo de algoritmos transforma las cadenas de caracteres en subconjuntos de cadenas de tamaño n y después compara el conjunto obtenido para una cadena con el conjunto obtenido para otra determinando así la similitud de las mismas en función del número de elementos que tengan en común. Por ejemplo la palabra *sacapuntas* para $n = 4$ se transforma en el conjunto $C = [saca, acap, capu, apun, punt, unta, ntas]$. Algunos ejemplos de estos algoritmos son los siguientes:

Algoritmo	Descripción
Índice de Jaccard	Este índice mide la similitud entre los elementos de dos conjuntos y se calcula mediante la fórmula: $$J(A,B) = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$$ donde el conjuto A es el conjunto de *q-grams* que forma una cadena de caracteres y el B el conjunto que forma la cadena con la que se va a comparar.

Coeficiente de Sorensen-Dice	Este índice es muy similar a Jaccard con la diferencia de que da más importancia a aquellos elementos que se encuentran en los dos conjuntos. El valor de similitud viene marcado por la fórmula: $$QS = \frac{2\,\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert}$$ donde el conjuto A es el conjunto de *q-grams* que forma una cadena de caracteres y el B el conjunto que forma la cadena con la que se va a comparar.
Similitud Coseno	Este se calcula realizando una transformación vectorial del conjunto de los *q-grams* que forman las cadenas de caracteres. Posteriormente se calcula el ángulo que forman los dos vectores (la cadena original y la que se pretende comparar) con la siguiente fórmula donde A A es el conjunto de *q-grams* que forma una cadena de caracteres y el B el conjunto que forma la cadena con la que se va a comparar: $$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2}\,\sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (2.1)$$ Esta da lugar a una medida de similitud, ya que si el valor de dicho ángulo es de 90° esto significará que las cadenas no comparten similitud alguna mientras que si es de 0° indicará que son exactamente iguales. El rango de valores entre 0° y 90° será por tanto [0, 1] tras aplicar el coseno sobre el ángulo.

Tabla 2.2. Algoritmos basados en la distancia q-grams

- **Basados en ontologías:** Este tipo de algoritmos se basan en la utilización de una ontología lingüística fuerte, de manera que son capaces de establecer un grado de similitud entre dos términos. Por ejemplo, si tenemos los términos "teléfono" y "ordenador" ambos resultan ser términos relacionados con la tecnología, por lo que estos tipos de algoritmos serán capaces de establecer una relación entre ambos y dar un valor de similitud en el rango [0, 1]. En esta línea se enmarca el artículo "Semantic Similarity Measures Applied to an Ontology for Human-Like Interaction" [10].

2.1.2.2. Problema de equivalencia

Cuando hablamos de la aproximación de cadenas desde el punto de vista de la resolución de un problema de equivalencia, nos encontramos esencialmente ante la situación en la que para determinar que una cadena o palabra es equivalente a otra, debe ser posible sustituirla por esa otra sin que esto afecte al contexto. Este tipo de problemas se presentan de distintas formas en función del tipo de modificación que se realice sobre la cadena.

- **Formateo de la cadena** : Este caso se da cuando se hace uso de mayúsculas y minúsculas, de caracteres especiales como guiones, puntos, comas, etc, el uso de los espacios para formatear texto... Por ejemplo, si tenemos la palabra *sacapuntas* podríamos formatearla de distintas maneras:

Tipos de formato
sacapuntas.
s a c a p u n t a s
saca puntas
saca-puntas
Saca Puntas

Tabla 2.3. Ejemplos de formateo de cadenas

  Una manera de abordar este problema es eliminar los espacios, signos de puntuación y caracteres especiales o transformar todos los caracteres de la cadena de texto a minúsculas o mayúsculas.

- **Variantes gramaticales** : Este caso se produce debido a que la misma palabra puede tener múltiples variantes debido al cambio de género, de número, los tiempos verbales... Por ejemplo, en el caso de *libro*, existen las palabras *librero*, *librería*... Principalmente hay dos tipos de algoritmos que se encargan de resolver este problema mediante la reducción del término en cuestión a su raiz.

  Normalmente los mecanismos que llevan a cabo la resolución de este problema reciben el nombre de *Algoritmos de Stemming* y se encargan de reducir una palabra a su raíz mediante el uso de unas reglas definidas. El algoritmo de *Stemming* más común es el algoritmo de Porter [11] propuesto en el año 1979 y que ha ido actualizándose y refinándose a lo largo de los años hasta encontrarse disponible en una gran parte de los lenguajes de programación más usados.

Palabra	Stemmed
cats	cat
weaksness	weak
required	requir
example	exampl

Tabla 2.4. Stem de términos

Por tanto, con el uso de este algoritmo podemos reducir una palabra a su raíz y se toma como equivalente otra palabra que tenga la misma raíz. La desventaja de este tipo de mecanismos es que son dependientes del idioma y por lo tanto no se pueden aplicar de forma genérica.

- **Variantes semióticas** : Este caso quizás es el más complejo dentro de la resolución del problema de equivalencia ya que para sustituir una palabra por otra ambas deben compartir el mismo significado, es decir, que el contexto en el que se encuentren no cambie debido al cambio de palabra. La razón de la complejidad de este caso no es otra que la necesidad de utilizar técnicas del lenguaje natural para poder reconocer la semántica de la cadena y así poder resolver el problema de equivalencia de forma correcta. Por ejemplo, el uso de *WordNet* para encontrar relaciones de sinonimia entre dos palabras y que de esta manera, una pueda sustituir a la otra.

2.1.2.3. Similitud vs Equivalencia

Como observamos entonces, el problema de equivalencia y de similitud son dos formas completamente distintas de abordar la técnica de aproximación de cadenas, no obstante no estamos limitados a escoger uno de ellos cuando nos encontramos ante la necesidad de utilizar la técnica de la aproximación de cadenas. Podemos decir que ambas maneras de enfocar el problema son complementarias. Esto quiere decir que podemos aplicar en primer lugar los mecanismos que resuelven el problema de equivalencia para posteriormente, abordar el problema de similitud de cadenas. De esta manera nos encontramos que la similitud de cadenas se realiza con unas cadenas que se han estandarizado, limpiado o ampliado con las técnicas de equivalencia por lo que podremos encontrar un mejor resultado al obtener la similitud.

2.2. Marco de aplicación

Las dos técnicas que se han explicado en el punto anterior, fueron, y siguen siendo hoy en día la teoría base que se toma como punto de partida a la hora de encontrar aplicaciones que faciliten ciertos ámbitos de la informática. Por ejemplo, en el caso de la teoría de conjuntos difusos surgieron diversas aplicaciones, pero la que más compete al trabajo aquí propuesto como ya se ha indicado en el punto *"Aplicación en sistemas gestores de datos"*, es el uso de la misma aplicado a las bases de datos. De esta manera, todas las operaciones que se podían realizar con una base de datos tradicional se fusionaron con la teoría de la incertidumbre y los conjuntos borrosos para poder tratar de una manera más humana la información almacenada.

En el caso de las técnicas de aproximación de cadenas, su punto fuerte dentro del campo de la informática recae principalmente en actividades como los sistemas correctores, detectores de plagio, recuperación y tratamiento de la información... En definitiva, todas aquellas tareas que requieran del procesamiento de textos para llevarlas a cabo. Cabe destacar que los sistemas que hacen uso de estas técnicas están íntimamente relacionadas con los gestores de almacenamiento de datos, puesto que la fuente de información que utilizan como recurso no es otra que una base de datos tradicional en la mayoría de los casos.

Por tanto, podemos observar como ambas técnicas han sido precursoras de aplicaciones que a día de hoy se siguen manteniendo, y que ambas se utilizan, o bien para enriquecer las propiedades de un sistema de almacenamiento de datos (como es el caso de las bases de datos difusas) o bien aumentar la funcionalidad de los mismos mediante las técnicas de aproximación de cadenas (como es el caso de la recuperación de la información de forma difusa). Es en este punto donde nos percatamos de que las bases de datos están íntimamente ligadas a estas dos técnicas, pero el avance de la tecnología es inevitable, y con el paso del tiempo nos hemos encontrado que las bases de datos clásicas tal y como se conocían hasta hace unos años han evolucionado para satisfacer otro tipo de almacenamiento: el que se genera debido a la gran cantidad de información de la que disponemos hoy día. Por tanto es de esperar que estas técnicas que son tan utilizadas y útiles en el ámbito de las bases de datos tradicionales, tengan su cabida en la nueva era de almacenamiento de la información.

2.2.1. Las bases de datos NoSQL

Hoy en día el auge de internet como plataforma servidora de aplicaciones (web, móvil, redes sociales...), ha producido que el volumen de datos que manejan dichas aplicaciones crezca exponencialmente, ya que cada vez son más los usuarios que demandan este tipo de tecnología. Dadas las grandes cantidades de información que se manejan, no es extraño que hayan surgido otros tipos de paradigmas que permitan almacenarla. Las llamadas bases de datos NoSQL (*Not only SQL*) son idóneas para esta situación, ya que permiten el almacenamiento masivo de la información para su posterior tratamiento y no tienen tantas

restricciones como las bases de datos tradicionales. Entre las principales características que ofrece esta tecnología en contraposición a una base de datos tradicional se encuentran las siguientes:

- Poseen mayor escalabilidad.
- Dan mejor rendimiento para la realización de operaciones.
- Poseen bajos costes de operación.
- El modelo de datos que utilizan es flexible.
- Ofrecen flexibilidad en el desarrollo y rapidez en la implantación.

Cabe destacar que dentro de las tecnologías NoSQL, surgen distintos paradigmas en función de los requerimientos de la aplicación a la hora de almacenar la información. Principalmente, se distinguen 4 paradigmas:

- Clave-Valor: Almacenan la información de una forma desestructurada consistente en una clave que tiene asociado un valor y son las más populares dentro de la tecnología del NoSQL debido a su simplicidad y por tratarse de bases de datos muy rápidas tanto para escritura como para lectura

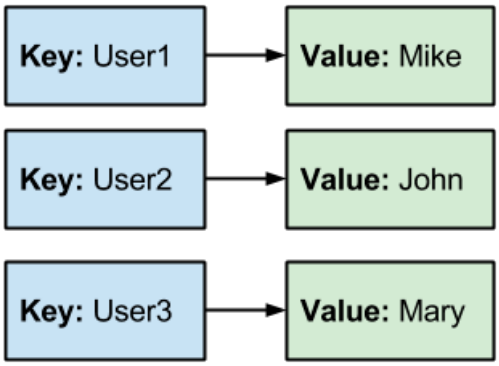

Fig. 2.5. Almacenamiento en el paradigma clave-valor

  Se usan sobre todo en aplicaciones que almacenan perfiles de usuario, la información del carrito de la compra de cualquier *e-commerce*... En definitiva, en sistemas que necesiten de información recuperable por una clave. Algunos ejemplos de este tipo de bases de datos NoSQL son Redis, Riak o Aerospike.

- Orientadas a Documentos: Son las más versátiles y se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían en bases de datos relacionales. Consisten en el uso del un estándar (JSON, BSON, XML y YAML) para generar el modelo de documento, de esta forma podemos gestionar información compleja de manera flexible.

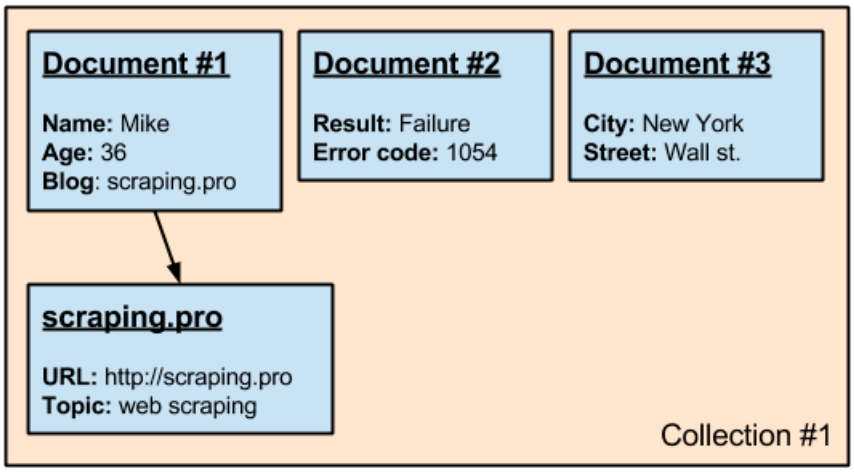

Fig. 2.6. Almacenamiento en el paradigma orientado a documentos

Algunos ejemplos de gestores que implementan este paradigma de almacenamiento son MongoDB, CouchDB y Elastic.

- Orientadas a Grafos: Consisten en la relación entre entidades y las conexiones que se realizan entre ellas. Son muy utilizadas cuando es necesaria una navegación más eficiente entre relaciones que la que ofrece el modelo relacional.

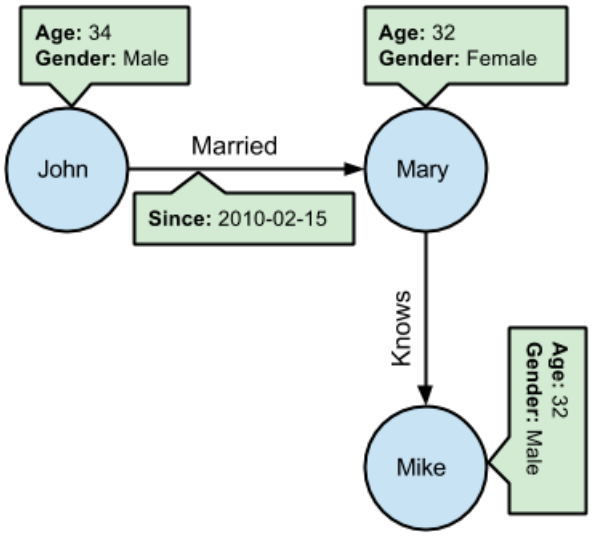

Fig. 2.7. Almacenamiento en el paradigma orientado a grafos

Por ejemplo, se utilizan mucho para almacenar toda la información referente a redes sociales, aunque se pueden aplicar a otros tipos de aplicaciones como catálogos, motores de recomendación... Algunos ejemplos de estas son Neo4j, Titan y OrientDB.

- Orientadas a Columnas: Estas son las más similares a las bases de datos tradicionales con la salvedad de que un registro puede contener un número cualquiera de

columnas, es decir, no tiene la limitación de las BBDD relacionales en las que se definen exactamente las columnas de una tabla. Son muy utilizadas cuando se requiere el almacenamiento de grandes cantidades de datos (del orden de PetaBytes) y se realizan muchas más operaciones de lectura que de escritura.

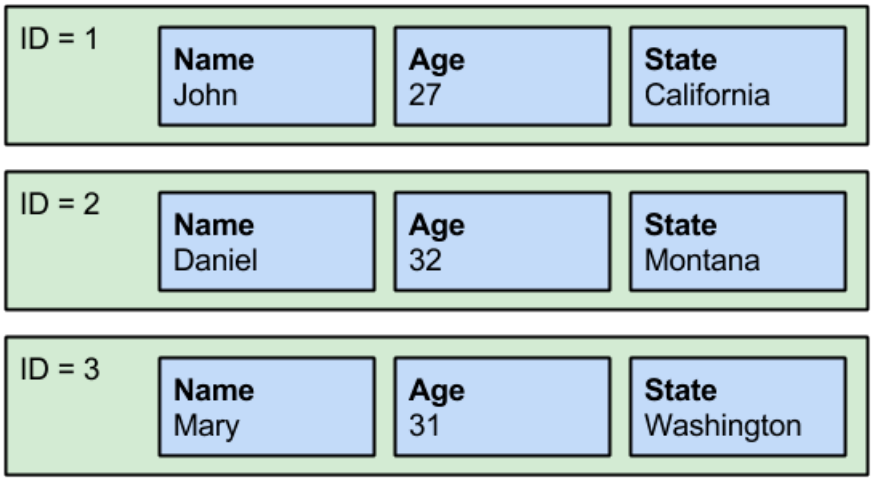

Fig. 2.8. Almacenamiento en el paradigma orientado a columnas

Algunos ejemplos de gestores que implementan este paradigma de almacenamiento son Cassandra y HBase.

Como podemos ver, cada uno de estos paradigmas de las bases de datos NoSQL posee una serie de características distintas que lo hace más indicado para almacenar cierto tipo de información en función del uso que se le vaya a dar a la misma. Pero a pesar de tratarse de sistemas muy completos y que facilitan el tratamiento de la información como se había conocido hasta el momento con las bases de datos relacionales, ¿Qué ocurre si se trata de una aplicación que hace uso de este tipo de bases de datos para almacenar hoteles y toda la información relacionada con ellos, y el usuario desea saber cuales de ellos son los más 'baratos'?. En este punto, surge la misma necesidad de tratar o recuperar la información de forma difusa debido a que el conocimiento que se almacena en este tipo de bases de datos, a pesar de ser más flexible, continúa siendo imperfecto en ocasiones.

Es por esto que surge la necesidad de introducir la posibilidad de recuperar información a partir de conceptos vagos o difusos también en las bases de datos NoSQL al igual que se introdujo en las bases de datos tradicionales.

2.2.1.1. Lógica difusa en NoSQL.

De forma nativa, la tecnología de almacenamiento NoQSL no dispone de la posibilidad de realizar operaciones difusas. No obstante, a diferencia de las bases de datos relacionales en las que se buscaba también un paradigma capaz de almacenar información recurriendo a la teoría completa de los conjuntos difusos con todas sus características,

en este caso, al ser el almacenamiento de la información menos estructurado, se persigue más la finalidad de poder trabajar con conceptos imprecisos.

En esta línea surge el trabajo de A. Castelltort y A. Laurent [12] en el que introducen la posibilidad de realizar consultas difusas sobre una base de datos NoSQL orientada a grafos, concretamente sobre Neo4j, realizando una extensión del lenguaje que se utiliza en dicho software (Cypher) para que pueda soportar este tipo de operaciones. En su trabajo, A. Castelltort y A. Laurent abogan por el uso de Neo4j y las bases de datos NoSQL orientadas a grafos debido a que es altamente escalable y en el momento en el que realizaron la investigación (2013) resultaba la más madura y completa. Construyen así un modelo de consulta difusa para Neo4j basado en f-SPARQL, un lenguaje de consulta difusa preparado para operar con ontologías, debido a la similitud que presenta una ontología con una base de datos orientada a grafos. Este modelo de consulta difusa, a grandes rasgos presenta la posibilidad de realizar consultas en Neo4j a nivel de:

- Las propiedades que se almacenan, lo que permite filtrar haciendo uso de etiquetas difusas. Por ejemplo por buscar hoteles 'baratos' u ordenar por los hoteles 'más cercanos' al centro de la ciudad.
- Los nodos, lo que permite recuperar nodos similares. Por ejemplo determinar si un hotel (un nodo) tiene unas características similares a otro hotel.
- Las relaciones, lo que permite añadir conocimiento difuso a las agregaciones (que en Neo4j se realizan a través de las relaciones entre nodos). Por ejemplo, determinar si un hotel es popular o no en función del número de clientes que lo visitan (el hotel es un nodo que posee relaciones con los nodos de clientes).

No obstante tiene un alto coste de mantenimiento, y se trata de un modelo restringido a Neo4j por lo que no podemos extenderlo a otro tipo de paradigmas de base de datos NoSQL.

Por esto surge también el trabajo de A. Belhadj Kacem y A. Grissa Touzi [13] en el que introducen la misma teoría de la importancia de poder realizar consultas difusas, con la diferencia de que lo realizan sobre un paradigma de las bases de datos NoSQL más extendido en la actualidad: las bases de datos NoSQL orientadas a documentos. Concretamente desarrollan su trabajo en MongoDB y proponen el modelo de *FMQL (Fuzzy Query Mongo Language)*, en el que desarrollan los siguientes elementos de consulta difusa para MongoDB:

- Etiquetas difusas: Permiten el uso de etiquetas como por ejemplo 'alto' o 'barato' para realizar la búsqueda o filtrado de los documentos.
- Comparadores difusos: Permiten el uso de comparadores como 'mucho mas que' o 'menos que' entre otros para filtrar el resultado de los documentos.

Estas consultas las llevan a cabo definiendo un traductor que se encarga de, dada una consulta con atributos difusos como los que se han mencionado en los dos puntos anteriores, realiza la conversión al lenguaje natural de MongoDB (MQL) que será el que se ejecute sobre la base de datos. Por ejemplo, si se trata de una etiqueta difusa 'joven', el traductor convertirá la consulta en la que especifica esta etiqueta en el lenguaje pertinente de MongoDB con los comparadores > y ⩽.

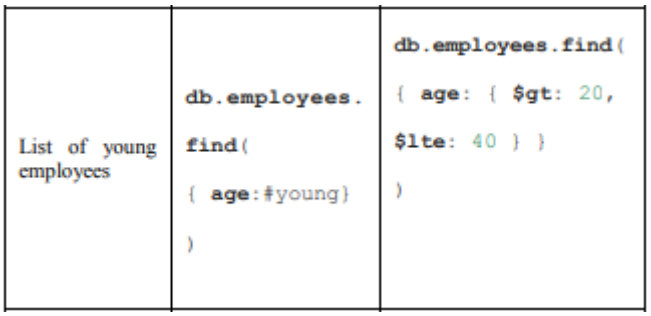

Fig. 2.9. Transformación de FMQL a MQL [13]

A parte de este traductor (que actúa también para el caso de los comparadores difusos), es necesario disponer de una base de conocimiento previamente almacenada en la base de datos para conocer realmente cual es el significado de la etiqueta 'joven'. Es decir, se almacenan los conjuntos difusos con su función de pertenencia para que el traductor sea capaz de determinar que lo que quiere decir cuando se indica la etiqueta 'joven' en una consulta FMQL, es que la edad se sitúe entre los valores 20 y 40.

Por tanto, vemos como la teoría de conjuntos introducida por Zadeh en 1964 se aplica a las bases de datos NoSQL al igual que fueron aplicadas en las bases de datos tradicionales.

2.2.1.2. Aproximación de cadenas en NoSQL.

En cuanto a la aproximación de cadenas, esta técnica si viene implementada en algunas soluciones de bases de datos NoSQL. Algunos ejemplos de ello son la búsqueda de texto que permiten realizar los gestores MongoDB o ElasticSearch. El funcionamiento es básicamente recuperar la información de forma difusa a partir de la comparación de la cadena de texto introducida con la información que se encuentra almacenada. No obstante, ninguno de ellos hace uso de las técnicas de *q-grams* por ejemplo, ya que ambos basan la determinación de la similitud de la cadena resultado con la cadena introducida en la consulta con algoritmos basados en la distancia de edición, generalmente usando el algoritmo de *Levenshtein* y un umbral de similitud determinado por el usuario. Además, este tipo de recuperación difusa son solamente aplicables a un campo o atributo de los objetos almacenados en la base de datos.

En definitiva, tanto la teoría de conjuntos difusos como la aproximación de cadenas, se ha usado en bases de datos NoSQL para ampliar las operaciones y/o características

básicas que ofrecen, y en esta línea de querer aumentar las herramientas ofrecidas por esta tecnología tan usada en los últimos años, tiene cabida nuestra propuesta.

2.3. Conclusiones al estado del arte

Así pues como se ha estudiado a lo largo de este capítulo, la teoría difusa es de gran importancia cuando se trata del tratamiento de la información, ya que esta no es siempre perfecta. Es por esto por lo que surgen bases de datos como GEFRED que implementan la lógica difusa para el tratamiento de los datos que almacena cuando estos no representan conocimiento perfecto.

Además de esta, existen otras técnicas difusas como la aproximación de cadenas, que comprende algoritmos cuyo principal objetivo es determinar como de parecidas son dos cadenas. Este tipo de técnica difusa por su parte, resulta de gran utilidad cuando queremos recuperar información sin tener un conocimiento previo de como se encuentra almacenada. Estas están quizá más extendidas que la aplicación de la lógica difusa, pero ambas son dos herramientas potentes que, unidas al tratamiento de la información, pueden ofrecer nuevas vías cuando nos encontramos ante el análisis o la recuperación de datos imperfectos.

Es por esto que resulta de vital importancia entonces, al igual que se aplican estas técnicas sobre base de datos tradicionales, aplicarlas sobre la nueva generación de gestores de bases de datos: Las bases de datos desestructuradas o NoSQL. Aunque estas ya implementan ciertas técnicas para la recuperación de información difusa, como se ha analizado en capítulos anteriores, no se trata de una vía de estudio muy explotada, con la que se cree, mejorarían algunos aspectos de estas bases de datos. Es por esto que, tras la exposición de las distintas técnicas existentes aplicadas a las bases de datos NoSQL, el presente trabajo versa sobre la ampliación de las operaciones básicas que estas ofrecen, explotando de mejor manera las técnicas difusas existentes.

3. DEFINICIÓN DEL PROYECTO

Esta sección se ha organizado comenzando por un análisis genérico para ir refinándolo a lo largo de los puntos que la componen. De esta manera, en primer lugar se establecen los objetivos, el marco socio-económico y regulador y se especifican los requisitos dando una primera visión de la situación que abordamos en este proyecto. Se continua con un estudio de viabilidad acerca de que paradigma de almacenamiento se puede utilizar para la realización de este proyecto teniendo en cuenta las diversas soluciones que propone el NoSQL, mencionadas anteriormente, así como un activo software que implemente dicho paradigma.

Posteriormente se definen las especificaciones del sistema, extrayendo los aspectos clave del problema que nos ocupa explicados en el primer punto. De esta manera se podrá proceder a darles una solución.

Finalmente, se establece el plan de pruebas a partir del estudio de la especificación del sistema y el entorno operacional escogido. Además se escoge un ciclo de vida que se seguirá a lo largo de todo el proyecto, así como la metodología a implementar. Para cerrar el capítulo se realiza una estimación temporal y dineraria del proyecto.

3.1. Establecimiento de objetivos

Como se ha mencionado en el punto *"Las bases de datos NoSQL"*, la gran cantidad de datos que se genera hoy en día debido al uso de aplicaciones móviles, web, redes sociales, etc y la necesidad de almacenar y acceder a esos datos de forma rápida y eficaz ha propiciado que la escalabilidad y el rendimiento sean dos propiedades obligatorias en un gestor de bases de datos. Esta situación fue la que dio lugar a la aparición de nuevos paradigmas de almacenamiento: las bases de datos NoSQL.

La característica principal de este nuevo paradigma de almacenamiento es el hecho de que NoSQL no posee un almacenamiento estructurado, es decir, no tiene una estructura de tabla fija a diferencia de como ocurre en las bases de datos tradicionales. Es por esto que la velocidad de recuperación y la escalabilidad de estas bases de datos es mayor, lo que las hace una herramienta perfecta para almacenar una gran cantidad de datos sin que el rendimiento en la recuperación se vea afectado.

No obstante, al igual que sucedía con las bases de datos relacionales, la información que se almacena en este tipo de bases de datos puede no ser perfecta y que sea necesario el uso de técnicas difusas para la recuperación y el tratamiento de este tipo de información. Como se ha comentado en los apartados *"Lógica difusa en NoSQL"* y *"Aproximación de cadenas en NoSQL"* ya se han implementado algunas técnicas difusas sobre las bases de datos NoSQL para dar soporte a esta situación, pero siempre desde el punto de vista de la

recuperación de información y la realización de consultas.

Por un lado, se dispone de los casos en los que se realiza una implementación ajena al gestor, es decir, una ampliación que no contemplan los gestores en su forma nativa lo cual puede resultar tedioso a la hora de hacer uso de dichas tecnologías. Si bien es cierto que implementaciones como el caso del uso de etiquetas y comparadores difusos en MongoDB y Neo4j, explicadas anteriormente, pueden resultar verdaderamente útiles cuando necesitamos recuperar información de forma imprecisa. Por ejemplo si se dispone de datos en los que se puntúan películas y series, este tipo de implementaciones permite realizar consultas del tipo "Las mejores películas de acción".

Por otro lado, aquellos casos que los gestores de bases de datos NoSQL si que contemplan, recae principalmente en la recuperación de los datos cuando se realiza una búsqueda imprecisa. Por ejemplo, utilizando el mismo caso en el que disponemos de datos sobre películas y series, si se introduce un nombre erróneo a la hora de realizar una búsqueda, el sistema es capaz de mostrar los resultados que están relacionados con la consulta que se ha realizado mediante el uso de técnicas de aproximación de cadenas. Esto también resulta útil, e incluso necesario, pero se sigue la misma línea de hacer uso de las técnicas difusas única y exclusivamente para recuperar la información. E incluso en este caso en el que las bases de datos NoSQL si implementan estas técnicas de forma nativa, se limita la recuperación de información de forma difusa únicamente a información que se encuentra almacenada como cadenas de texto.

La pregunta que surge entonces es si es posible mediante el uso de las técnicas difusas y gracias a la relajación del modelo de datos que presentan las bases de datos NoSQL (a diferencia de las bases de datos relacionales que son más restrictivas) aumentar las posibilidades que ofrecen estos gestores. Actualmente el uso de estas técnicas se limita únicamente a la recuperación de la información que se encuentra almacenada, bien sea mediante las herramientas ofrecidas lógica difusa propuesta por Zadeh o mediante las técnicas de aproximación de cadenas anteriormente mencionadas.

Es por esto que la línea que sigue este proyecto versa sobre la ampliación de las operaciones que las bases de datos NoSQL ponen a nuestra disposición mediante el uso de las técnicas difusas y aplicar estas técnicas al resto de operaciones disponibles y no solamente a la recuperación de información.

3.2. Entorno socio-económico

Una vez se ha llegado a este punto, se puede mencionar la importancia de realizar este proyecto debido a que no existe en la actualidad una explotación de este tipo de herramientas, lo cual puede repercutir positivamente en el uso de las bases de datos NoSQL. Además el hecho de que no sea una aplicación muy usual sobre las bases de datos NoSQL lo hace un producto atractivo si se consigue que el funcionamiento del mismo sea aceptable, es decir, que sea capaz de realizar las tareas de análisis de los datos utilizando técnicas difusas, y que estos resultados sean mejores que los obtenidos mediante los métodos utilizados hasta el momento.

Además, como ya se ha mencionado en la introducción, y posteriormente a lo largo del estado del arte, estas bases de datos han adquirido en la actualidad una función muy importante, ya que son el principal motor impulsor del análisis cuando hablamos de grandes cantidades de datos. Por tanto, disponer de una herramienta que nos pueda ayudar a la hora de analizar dichos datos y a identificar de forma más rápida y precisa, por ejemplo las necesidades que tiene un usuario, puede repercutir en beneficios económicos para aquél que sea el afortunado de utilizar dicha tecnología y poder ofrecerle a ese usuario aquello que necesita sin gastar mucho tiempo o dinero. Si esto lo se extrapola a hacerlo con todos los usuarios que sean clientes de una empresa en cuestión, seguramente se produzca un aumento de los beneficios, ya que el gasto que se desempeña es menor al contar con las herramientas difusas que son capaces de realizar satisfactoriamente la asociación producto-usuario, y un mayor beneficio debido a que las probabilidades de que el usuario en cuestión realice la compra son mayores.

Por tanto, se produce la situación en la que un producto, al no estar explotado en la industria actual, presenta posibilidades de perfilarse como un software necesario a la hora de utilizar la tecnología NoSQL y además, mejorar aquellas áreas donde actualmente se están aplicando análisis de datos, ya que permitiría realizar dichos análisis de una forma más “humana” al utilizar un modelo computacional que no está basado en la lógica clásica.

3.3. Marco regulador

En este punto se establece el marco regulador que atañe al trabajo aquí expuesto. Es importante tratar con especial cuidado esta sección puesto que se trata del entorno legal que afecta al desarrollo del trabajo.

Puesto que la solución final que se va a aportar es un sistema funcional que extienda las operaciones difusas de una base de datos NoSQL, se debe tener en cuenta especialmente la licencia bajo la que se libera dicho sistema. Este punto afecta principalmente al uso que se le podrá dar al sistema más adelante, ya que no es lo mismo que posea una licencia de carácter libre, con propiedad intelectual o privada. La primera nos permitiría el uso indiscriminado del sistema independientemente del fin para el que se vaya a utilizar, la segunda no permitiría su uso con fines lucrativos, solamente para fines académicos o de investigación, y con la última se debe adquirir una licencia del sistema por la que se tendría pagar previamente a su utilización. Esta situación, en la que se dispone de tres vías diferentes en las que enmarcar el sistema, se debe analizar y posteriormente escoger la que mejor de adapte al proyecto.

Dicho esto se descarta de inmediato la opción de establecer una licencia privativa o comercial para el sistema final. Las razones de que esto sea así no son otras que la necesidad de utilizar en el proyecto soluciones de terceros, ya que para el desarrollo, y como se explicará con más detalle en capítulos posteriores, es necesaria la implementación de varios algoritmos los cuales no son de nuestra autoría, si no que se reproducirán gracias a librerías y módulos previamente desarrollados por otras entidades u organizaciones. Además se le debe sumar el uso de herramientas que se utilicen para llevar a cabo el desarrollo. Todas ellas serán de origen libre (licencias GNU y MIT), es decir, que permitan la utilización del código sin tener que realizar un desembolso a su dueño intelectual o solicitarle una autorización de uso. Por esto el sistema deberá estar sujeto al menos a estas mismas restricciones, descartando entonces que este se acoja a licencias de tipo Shareware, privativas o comerciales [14], en las que directamente se vende el servicio que oferta el sistema o se necesita de una autorización para su uso.

Entre las dos opciones restantes: utilizar una licencia totalmente libre que permita el uso del sistema para cualquier propósito, o utilizar una licencia que disponga de copyleft, es decir, que solamente permita el libre uso del sistema para investigación y desarrollos de software que serán también libres, se escoge esta última al resultar la más acorde con el trabajo que se está exponiendo. Cómo se adelantaba en el marco socio-económico, está solución puede suponer el comienzo del desarrollo unas herramientas que podrían completarse en un futuro y ser de gran utilidad para el análisis de datos (el punto fuerte para el que se requieren las bases de datos NoSQL). Por ello, el sistema estará enmarcado con una licencia de software libre con copyleft, es decir, el software que haga uso de nuestro sistema deberá ser también libre.

Concretamente se ha seleccionado la licencia GNU GPLv3 que incorpora la licencia

de carácter libre con copyleft [15], de manera que los usuarios que utilicen el sistema deberán de desarrollar también software libre. A parte de las razones que se han expuesto con anterioridad sobre la elección de esta licencia, además dicha elección atiende a razones éticas, ya que de esta manera se fomenta la difusión del conocimiento sin trabas.

3.4. Establecimiento de requisitos

Una vez expuesto el problema que se pretende abordar, se establecen los requisitos que el enmarcan el proyecto y los límites del trabajo. En otras palabras, se expone que va a a poder realizar la solución final y que no.

Como producto de las reuniones que se mantuvieron acerca del trabajo posteriores a la investigación y redacción del estado del arte, se acotaron los caminos que se podían seguir. Hasta ahora, se ha hablado sobre aumentar las operaciones base ofrecidas por un gestor de base de datos NoSQL haciendo uso de técnicas difusas. Puesto que se trata de un campo inexplorado en el que la investigación que se realizó previamente solamente arrojó trabajos en los que se aplicaban técnicas difusas para la recuperación de información, se decide comenzar por lo más básico: las operaciones CRUD (del acrónimo inglés Create, Read, Update y Delete). Por tanto este trabajo tendrá cabida en hacer uso de las técnicas difusas mencionadas en el estado del arte y extraer nuevas funcionalidades directamente relacionadas con las operaciones de inserción, borrado, actualización y recuperación (pese a que este último punto es el que más se había explotado en soluciones previas a las que este trabajo expone).

Se explicará más adelante como se ha realizado este enfoque, pero a grandes rasgos se puede definir como realizar la inserción, borrado y recuperación de forma difusa.

Todas estas operaciones se basarán en comparar la similitud de los registros, que es la unidad de almacenamiento de cualquier base de datos NoSQL. Actualmente, lo más parecido a esto es recuperar registros en los que un atributo o varios que almacenan un String, sean similares a los que se introducen en la consulta que se especifica. Por ejemplo, en una base de datos que recoja artículos, que recupere todos los registros cuyo nombre de artículo sea similar a *Blackberry* pudiendo recuperar otros como *blueberry* (arándano en inglés) o *raspberry*. Con este proyecto se pretende lograr la comparación entre registros completos (independientemente de los atributos y tipos de dato que contengan los mismos), y dados dos registros establecer un valor de similitud entre ellos mediante la utilización de las técnicas expuestas en el apartado del estado del arte. Como se realizará esto de forma funcional se expondrá más adelante, pero en rasgos generales esta es la finalidad que se pretende conseguir.

Por supuesto, todas estas funcionalidades deben poder aplicarse a cualquier base de datos NoSQL, pero por comodidad para este proyecto (y debido a que excede el tiempo estipulado para realizar el trabajo) el trabajo se centrará únicamente en un tipo de estas bases de datos, cuya elección se realizará de forma detallada en el punto *"Viabilidad del sistema"*.

Como ya se ha dicho, a través de la comparación de los registros en función de su similitud se llevará a cabo la ampliación de características de los siguientes métodos:

- Inserción difusa de registros: Previamente a la inserción se comprueba si existe un

registro similar.

- Borrado difuso de registros: Se realiza una limpieza de la base de datos, borrando los registros similares que podríamos tomar como información duplicada.
- Recuperación difusa de registros: Se recuperan todos los registros similares a uno dado.

Como se habrá notado, no se incluye aquí la operación de actualización, escogiendo únicamente la creación y el borrado como operaciones "nuevas" y ampliando la recuperación difusa ofrecida por las bases de datos NoSQL de forma nativa a nivel de registro en lugar de atributo. Esto se debe simplemente a que se ha determinado que el borrado, la inserción y la recuperación difusa pueden ser herramientas potentes para el uso en las bases de datos. Sin embargo, en el caso de una actualización difusa, siguiendo la funcionalidad de las otras operaciones, se estarían actualizando todos los registros que fuesen similares a la condición introducida. Teniendo en cuenta que la comparación de similitud se pretende realizar a nivel de registro, cabe esperar que al actualizar los registros que sean similares a la condición dada, se propicie que estos sean más similares aun entre sí, o incluso en el peor de los casos, la aparición de registros duplicados. Es por esto que no se ha posibilitado la expansión difusa de esta operación.

A partir de aquí nacen los requisitos funcionales que conforman la que será la solución final con el objetivo de establecer los límites del trabajo. Los requisitos básicos del sistema son los siguientes:

- **RF-01** : El sistema debe ser compatible con un gestror de almacenamiento masivo de tipo NoSQL.
- **RF-02** : El sistema debe ser transparente al usuario que lo utiliza.
- **RF-03** : El sistema debe proveer de las funciones necesarias para que el usuario final haga uso de las operaciones expandidas sin importar la naturaleza de los datos.
- **RF-04** : El sistema deberá ser capaz de comparar el grado de similitud de un registro dado con otro, independientemente de los tipos de dato que contengan o la estructura del mismo.
- **RF-05** : El sistema debe implementar las técnicas y algoritmos difusas necesarias para calcular la similitud entre registros.
- **RF-06** : El sistema deberá permitir la inserción de registros difusa.
- **RF-07** : El sistema deberá permitir el borrado difuso de registros.
- **RF-08** :El sistema deberá permitir las consultas difusas de registros.

Por otro lado, y como ya se ha comentado se detallan también algunos aspectos que no contempla este trabajo:

- La actualización difusa de registros.
- La recuperación de documentos a partir de la consulta de un atributo de forma difusa (como ya viene implementado en las bases de datos NoSQL de forma nativa para el caso de las cadenas de caracteres). Es decir, la recuperación se realiza a nivel de registro única y exclusivamente.

En este caso no se han asociado con un identificador puesto que el sistema final no va a disponer de dichas funciones.

3.5. Análisis de viabilidad y selección de herramientas

En este punto se procede a realizar un análisis de viabilidad, es decir, sobre que casos concretos podemos aplicar los objetivos que se han determinado en este proyecto. Además una vez realizado dicho análisis, se estudian las tecnologías que se engloban dentro del mismo y sobre las que se efectuará el desarrollo final del sistema para poder escoger con criterio una de ellas.

3.5.1. Viabilidad del sistema

Para llevar a cabo la tarea de la ampliación de las herramientas de una base de datos NoSQL mediante las técnicas difusas, en primer lugar se debe barajar escoger una de las muchas opciones que se encuentran en el mercado y que ofrecen este tipo de paradigma de almacenamiento.

Actualmente, existe gran cantidad de software que implementa la tecnología NoSQL, pero en primer lugar cabe destacar que la utilización de esta solución en lugar de las bases de datos relacionales clásicas debe producirse cuando:

- El volumen de datos crece muy rápido.
- La escalabilidad que poseen las bases de datos relacionales no es suficiente tanto a nivel técnico como en costes.
- El sistema tiene picos muy grandes de uso por parte de los usuarios.
- El esquema de la base de datos no es homogéneo.

Por tanto si se pretende utilizar una base de datos NoSQL pero no se cumplen estos requisitos, quizá esta elección no sea la más adecuada y sea suficiente con una base de datos relacional. La razón de que en este trabajo se haya elegido la implementación de una ampliación de las herramientas que estos gestores proporcionan en lugar de en un gestor relacional tradicional, ha sido principalmente impulsada por la necesidad; en primer lugar porque el uso de las técnicas difusas en las bases de datos tradicionales está mucho más estudiado y se han realizado muchas más investigaciones sobre el mismo, y en segundo lugar y más importante, el gran impulso que están teniendo en la actualidad este tipo de bases de datos como principal contenedor del BigData.

El uso de las técnicas difusas sobre un gestor NoSQL, repercute directamente en las posibilidades que nos ofrecen para tratar los datos y poder de esta manera analizar de mejor forma el BigData, que no es más que un gran conjunto de datos no estructurados generados por las tecnologías modernas (redes sociales, dispositivos GPS, vehículos, búsquedas realizadas en internet...) y que tratados de la manera correcta pueden ayudar a las organizaciones y empresas a identificar nuevas oportunidades.

Es por esto que cobra importancia la ampliación de las características que ofrece una base de datos NoSQL, además de ser una de las razones por las que se ha decidido orientar la ampliación de características para este tipo de gestores de base de datos y no para un gestor tradicional. Además también han influido en la selección de esta tecnología para la realización de este proyecto, el hecho de que las bases de datos tradicionales ya contaban con su implementación difusa como se estudia en el punto *"Aplicación en sistemas gestores de datos"* y que NoSQl se trata de una tecnología relativamente nueva que abre el camino de nuevas posibilidades en el mundo del almacenamiento y gestión de información.

Dicho esto, queda clara la decisión de realizar el proyecto orientado a bases de datos no relacionales, pero al igual que explicamos en el punto *"Las bases de datos NoSQL"* existen cuatro paradigmas de almacenamiento distintos dentro de esta tecnología: Orientado a grafos, orientado a columnas, orientado a documentos y de clave-valor. Cada una de ellos ofrece distintas posibilidades, y a pesar de que la ampliación de características gracias a técnicas difusas podría aplicarse sobre cualquiera de los paradigmas, se ha decidido centrar el estudio únicamente en uno de ellos a la hora de desarrollar este trabajo.

Concretamente, el paradigma que se ha escogido ha sido el orientado a documentos. La elección de este paradigma se debe principalmente a su gran versatilidad y su funcionamiento simple gracias al almacenamiento de la información siguiendo un formato estándar. Se puede decir que dentro de los cuatro paradigmas principales existentes en el NoSQL este es el que se encuentra más estandarizado en cuanto al almacenamiento de la información y esta no se almacena de forma tan desestructurada, lo cual facilita el trabajo a la hora de tratarla y aplicar sobre ellas las técnicas difusas pertinentes explicadas más adelante, para lograr así el objetivo de este trabajo.

No obstante, a pesar de haber escogido este paradigma aun se debe analizar la tecnología que lo implementa, puesto que no existe un único software que trabaje con este tipo de almacenamiento en las bases de datos NoSQL. Entre los modelos más usados destacan principalmente dos que se estudian a continuación para poder escoger uno de ellos: MongoDB y CouchDB.

3.5.1.1. CouchDB

Se trata de una base de datos de código abierto que tiene como principales características la facilidad de uso y ser "una base de datos que asume la web de manera completa"[16].

CouchDB hace uso de la estructura JSON para almacenar los datos llamados documentos. La estructura de este se puede definir como se ve en la imagen:

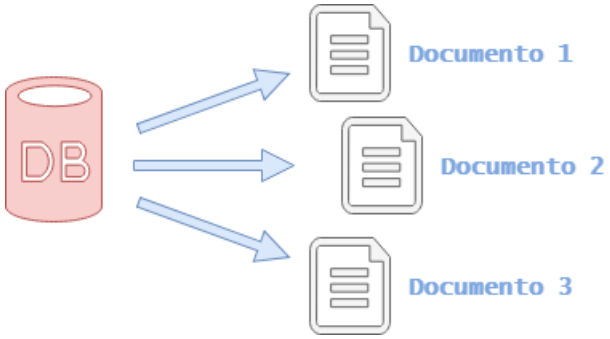

Fig. 3.1. Estructura de almacenamiento de CouchDB

Todos los documentos se almacenan directamente en la base de datos y cada uno de ellos está compuesto por un determinado número de claves que poseen un valor.

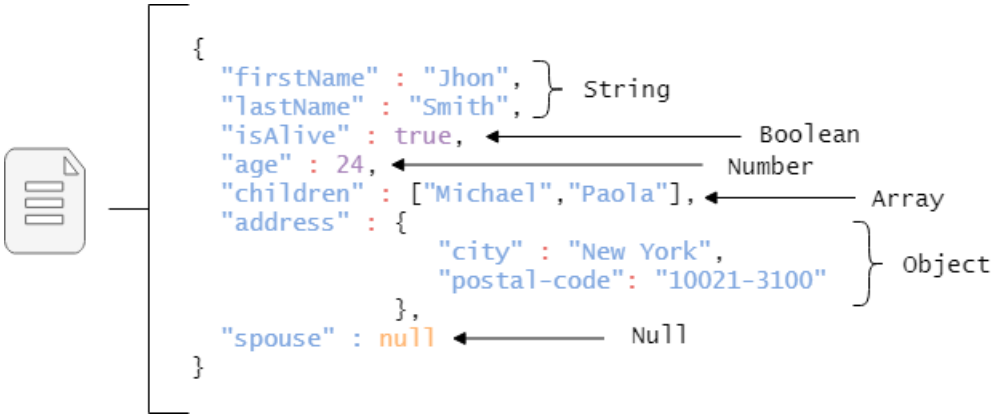

Fig. 3.2. Formato de un documento JSON

Como se observa en la imagen estos valores pueden representarse con cualquier tipo de dato que está permitido en el lenguaje JavaScript: Strings, números, objetos (de tipo JSON, estos serán objetos anidados), arrays, booleanos y valores *null*. Cabe destacar que todos los documentos almacenados en la base de datos tienen un identificador único y no requieren de un esquema determinado, es decir, en la base de datos se pueden almacenar documentos con estructuras totalmente dispares.

Para acceder a los documentos almacenados en la base de datos, CouchDB ofrece una estructura de consulta a través de HTTP como una API REST. Todos los documentos almacenados en la base de datos pueden ser recuperdos o transformados a través de los métodos HTTP: POST, GET, PUT y DELETE para realizar las operaciones CRUD (Create, Read, Update y Delete). Para utilizar estos métodos y poder interactuar con la base de datos se debe hacer uso de los métodos HTTP a través de una librería como cURL, un intérprete de comandos pensado para transmitir archivos y que soporta el protocolo HTTP entre otros [17].

3.5.1.2. MongoDB

Se trata también de una base de datos de código abierto pero no está tan orientada al desarrollo de aplicaciones web como si lo hace CouchDB.

MongoDB hace uso de una estructura distinta a CouchDB la hora de almacenar los documentos. A continuación se muestra un diagrama de como realiza dicho almacenamiento:

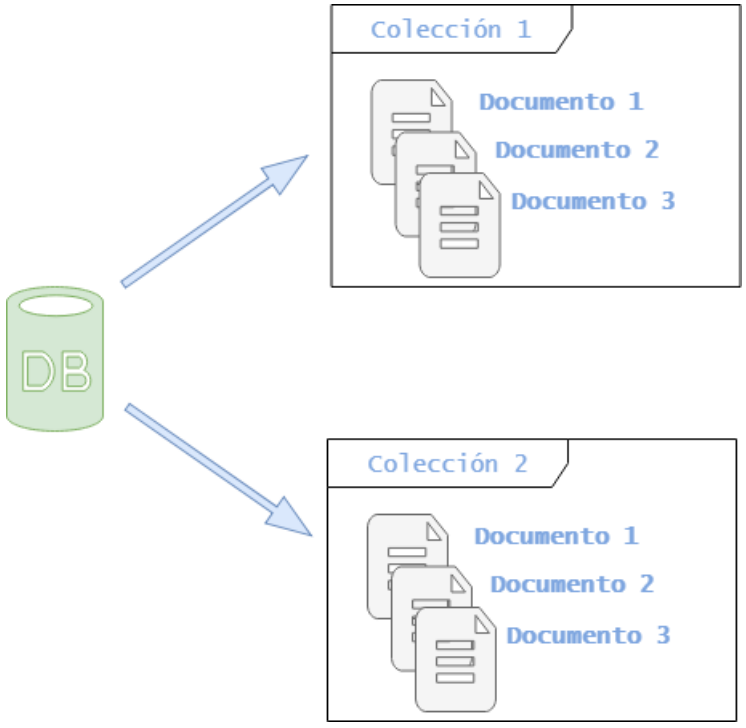

Fig. 3.3. Estructura de almacenamiento de MongoDB

Como se observa en la imagen, MongoDB no almacena directamente los documentos en la base de datos. En lugar de ello crea un elemento intermedio que reciben el nombre de colecciones, y es aquí donde se almacenan los documentos. Por tanto, una base de datos en MongoDB está compuesta por colecciones que a su vez contiene los documentos. Al igual que ocurre con CouchDB, los documentos pueden contener estructuras diferentes dentro de una misma colección, si bien es cierto que normalmente todos los documentos alojados en una colección comparten la estructura, e incluso en las últimas actualizaciones de MongoDB, existe la posibilidad de predefinir una estructura para una colección.

En cuanto a como se almacena la información dentro de los documentos, se hace uso de una estructura similar a la de JSON con ligeras modificaciones: utiliza un formato propio que se denomina BSON y que no es más que la estructura del JSON con algunos añadidos y en formato binario, lo que hace que sea más eficiente que este a la hora del almacenamiento y la recuperación de los documentos.

Fig. 3.4. Formato de un documento BSON

Como se observa en la figura anterior, el formato BSON es prácticamente el mismo que el formato JSON con el añadido de las fechas, y el objeto identificador que MongoDB utiliza para identificar los documentos almacenados en una colección.

Para acceder a la información, MongoDB hace uso de un lenguaje de consultas propio muy similar al lenguaje Javascript, además que permite la ejecución de código en este lenguaje a través de la consola de comandos que proporciona.

3.5.1.3. Comparativa y conclusiones.

En los puntos anteriores se ha expuesto las características más notables que intervienen a la hora de escoger una herramienta u otra para el desarrollo del estudio que se realiza en este trabajo. No obstante, conviene destacar que estas no son las únicas diferencias entre los dos gestores, poseen otras [18] que podrían resultar determinantes a la hora de decidir cual de ellos usar en otra circunstancia distinta a las que nos ocupa.

Dicho esto, las características de cada uno de los gestores de forma resumida son las siguientes:

	
Los documentos se almacenan directamente en la base de datos, pudiendo alojar una misma base de datos cualquier estructura de documento.	Los documentos se almacenan en colecciones, y estas en la base de datos. Las colecciones pueden almacenar cualquier tipo de estructura de documento, pero es más usual que los documentos almacenados compartan la estructura entre ellos.
El formato utilizado para almacenar los documentos es JSON.	El formato utilizado para almacenar documentos es BSON.
Las consultas se realizan mediante una API REST haciendo uso de los métodos que provee el protocolo HTTP.	Las consultas se realizan mediante un lenguaje propio similar a javascript y que permite la integración de este con el entorno de MongoDB,

Tabla 3.1. Comparativa de MongoDB y CouchDB

Conociendo las características de un gestor y otro, es el momento de escoger por cual se utilizará en el desarrollo del proyecto que nos ocupa.

Comenzando por el primer punto, el hecho de que MongoDB disponga del elemento intermedio de las colecciones y que prácticamente se tiene la certeza de que los documentos que se almacenan en una colección suelen compartir una estructura similar entre sí, hace que sea mucho más cómodo a la hora de tratar la información, ya que esta se encontrará más estructurada. Por ejemplo, en una base de datos en la que se quiera introducir información sobre películas y actores, en CouchDB de forma obligada se debe, o bien a crear dos bases de datos para almacenar los documentos referentes a películas en una y los referentes a actores en otra, o bien almacenarlos todos juntos. Mientras tanto, en MongoDB se puede crear una única base de datos con dos colecciones distintas, lo que da mucha más libertad a la hora de estructurar la información (a pesar de que esta tenga un carácter desestructurado).

En segundo lugar, el formato del documento no es muy relevante a la hora de desarrollar este trabajo. Solamente interfiere en la manera que se va a tener de tratar los valores que se almacenan en el documento, ya que el formato BSON de MongoDB posee más tipos de dato que el de CouchDB. Si bien es cierto que el hecho de que MongoDB utilice el formato BSON hace que las aplicaciones que se desarrollan con esta tecnología gocen de una velocidad algo mayor que cuando se utiliza un formato en JSON.

En tercer lugar, y quizás el más significativo a la hora de la elección por una de estas

dos tecnologías es como utilizarlas. En el caso de CouchDB si no se poseen los conocimientos necesarios se hace más difícil poder utilizarlo ya que se debe conocer los métodos que provee HTTP y conocer como realizar las llamadas para obtener la información de la base de datos. Sin embargo en el caso de MongoDB, se facilita la realización de consultas mediante un lenguaje que resulta más amigable y conocido por el equipo de desarrollo en este caso.

Finalmente destacar que a pesar de que ambos gestores son conocidos en el mundo de la informática, MongoDB goza de más popularidad entre los desarrolladores como muestra el siguiente gráfico.

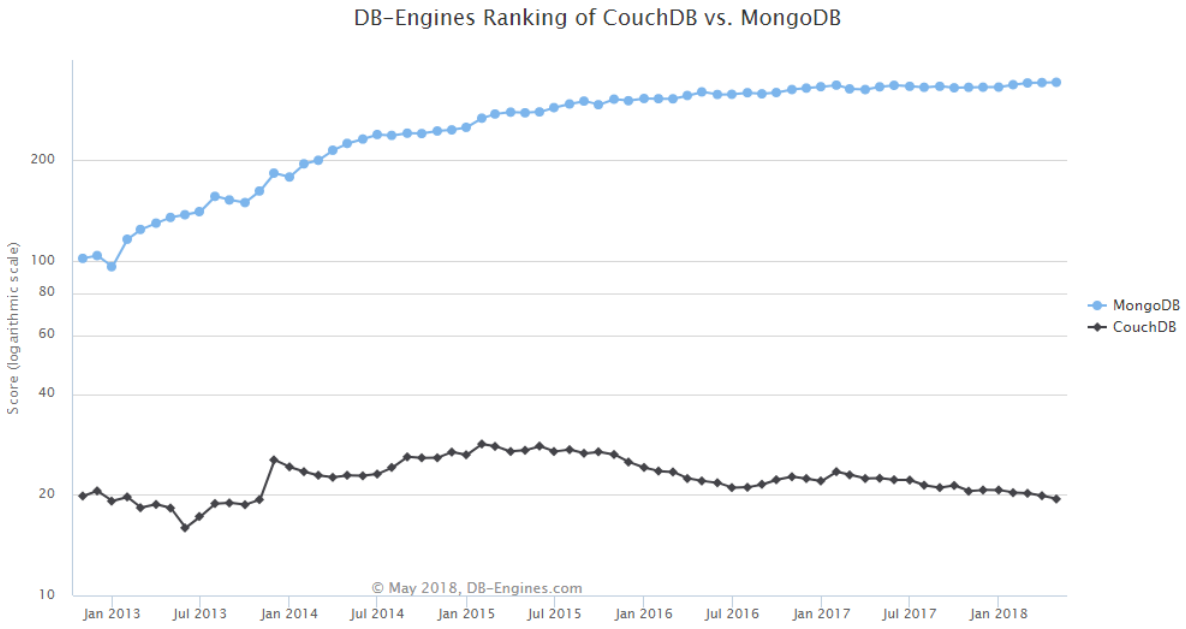

Fig. 3.5. Puntuación de MongoDB vs CouchDB [19]

En azul MongoDB y en negro CouchDB, podemos apreciar fácilmente que desde el año 2013 hasta el día de hoy el gestor MongoDB ha gozado de más popularidad. Dicha popularidad está basada en el número de veces que se nombra la tecnología en redes sociales o páginas web, las consultas que se realizan sobre ella en páginas especializadas, el interés general y el número de ofertas de trabajo y perfiles profesionales que hacen uso de ella. Por tanto, realizar este proyecto en MongoDB prácticamente asegura que va a resultar más útil que si lo realizamos para CouchDB.

Todo esto sumado a que el equipo desarrollador encargado de llevar a cabo el proyecto que nos ocupa posee conocimientos previos sobre MongoDB y el lenguaje de programación JavaScript, hace que la decisión final sobre la tecnología que se utiliza para realizar este trabajo sea MongoDB.

3.5.2. Establecimiento de la tecnología

Anteriormente, en el punto *"Viabilidad del sistema"* de esta misma sección se realiza un estudio sobre cual es la tecnología más adecuada para llevar a cabo el proyecto. Fruto de este análisis previo se extrae que MongoDB será el hilo conductor del desarrollo de nuestro trabajo.

Puesto que ya se ha decidido cual es el alcance del problema que nos ocupa y determinado las funcionalidades que va a recoger el sistema, es obligatorio, dado el software que se ha escogido, proponer alternativas de posibles implementaciones en las que desarrollar este proyecto teniendo en cuenta que el software escogido ha sido MongoDB y este permite implementaciones en JavaScript al ser un intérprete de dicho lenguaje.

Si bien esta última no es una condición obligatoria, puesto que la realidad es que existen innumerables módulos o librerías que permiten realizar operaciones en MongoDB desde distintos lenguajes de programación, si que es la más cómoda. Esto ocurre debido a que la propia consola que ejecuta MongoDB es un intérprete del lenguaje JavaScript, por lo que a la hora de realizar pequeñas pruebas o comprobaciones cuando se realice el desarrollo, será mucho más sencillo realizarlas directamente en la consola (o copiando el código en ella) antes que ejecutar un script en un determinado lenguaje de programación en el que previamente deberemos descargar las dependencias necesarias (el módulo pertinente que permita la conexión con MongoDB) y configurar la conexión a la base de datos si procede.

Dicho esto, se procede a analizar las distintas opciones en cuanto a la arquitectura que se puede utilizar para implementar el sistema, recordando que se realiza esta elección de cara a la solución final puesto que las dos primeras fases no necesitan de una arquitectura perfectamente definida para llevar a cabo su desarrollo. Por tanto desde este enfoque, utilizando como herramientas básicas JavaScript + MongoDB nos planteamos las tres alternativas que se aprecian en la imagen:

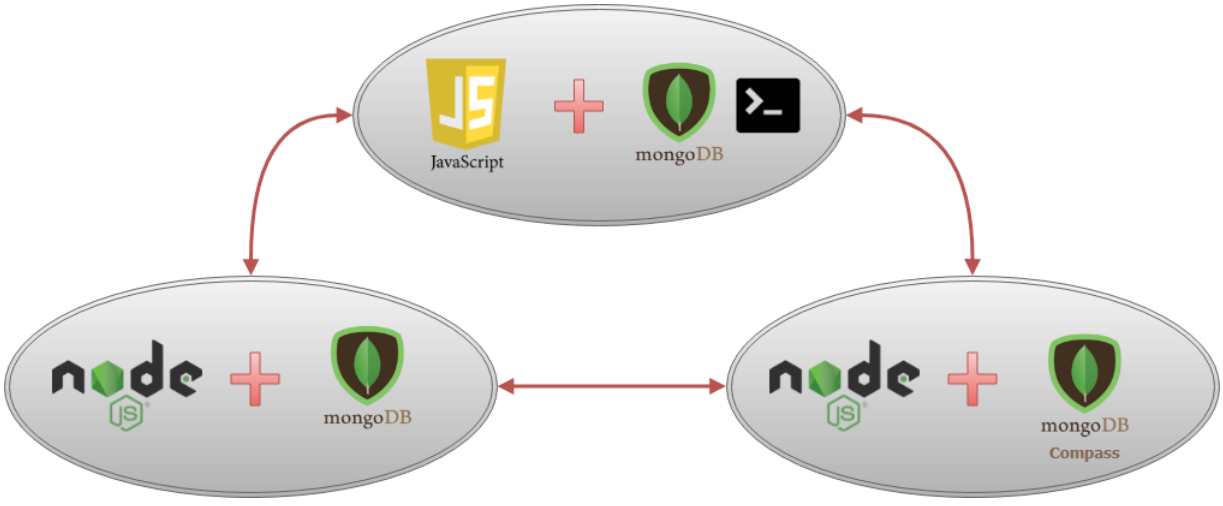

Fig. 3.6. Alternativas tecnológicas

- Utilizar JavaScript + MongoDB sin ningún añadido más e integrar el código implementado en la propia consola de MongoDB para que las funcionalidades nuevas estén disponibles como llamadas a funciones al ejecutar una instancia de conexión.
- Utilizar JavaScript en el entorno de ejecución Node.js creando un módulo que sea capaz de conectarse a una base de datos de MongoDB y tenga disponibles las nuevas funcionalidades a modo de API.
- Utilizar el entorno de ejecución de JavaScript Node.js en su integración con la herramienta Compass de MongoDB para que las funcionalidades estén disponibles a través de una interfaz gráfica.

3.5.2.1. Javascript + MongoDB

Esta alternativa hace uso del lenguaje JavaScript sin integrarlo con ningún tipo de framework. Simplemente se desarrolla el código y se ejecuta en la consola de MongoDB al ser esta un intérprete de dicho lenguaje. Es la alternativa más sencilla puesto que no es necesario la instalación y configuración de software adicional a MongoDB.

La idea de esta alternativa parte de desarrollar las funciones necesarias en JavaScript y ejecutarlas directamente en la consola, para lo cual deben cargarse en MongoDB previamente. Dado que copiar y pegar el código del desarrollo completo en la consola, o cargar el archivo donde se encuentra el código fuente por linea de comandos resulta muy tedioso se opta por integrar directamente dicho código con MongoDB para que cada vez que este inicie estén disponibles todas las funciones. Esto podemos realizarlo de dos formas distintas:

- Cargando las funciones en un fichero localizado en la carpeta de instalación de MongoDB. Este fichero, llamado *.mongorc.js* se encuentra vacío inicialmente, y todas las funciones y código que se introduzcan en el mismo se cargarán al iniciar una conexión a la base de datos [20]. No obstante, este método tiene varios inconvenientes:
  - El más importante es que si se desarrolla el sistema final de esta manera, para el correcto funcionamiento del sistema, el usuario se ve en la obligación de sustituir su archivo *.mongorc.js* con la consecuente pérdida de sus funciones (si este hace uso del mismo archivo). Para que mantuviese las funciones que se localizasen en el fichero (si las hubiera) sería el propio usuario el que debería integrar su código con el desarrollado en este proyecto, ya que MongoDB no ofrece la posibilidad de cargar varios ficheros distintos al inicio de la conexión y este es el que marca como predefinido.
  - En el caso de que el software de MongoDB no esté instalado en una máquina accesible se hace muy difícil la instalación del sistema, puesto que debería introducirse el fichero *.mongorc.js* en el directorio de instalación localizado

en el servidor, y en caso de no tener acceso al mismo, no se podría instalar el sistema. Además, las técnicas de sharding que permite MongoDB (tener los datos diseminados en varios servidores) hace que si se busca utilizar el sistema se deba replicar el archivo e introducirlo en cada uno de los servidores.

- Por último, a la hora de desarrollar resulta muy tedioso tener que utilizar el mismo fichero para implementar toda la funcionalidad del sistema. A pesar de que el desarrollo de este proyecto no contempla una extensa implementación con miles de líneas de código, si que resulta más cómodo poder estructurar el código en distintos archivos, además de que el mantenimiento es más sencillo de esta manera.

- Cargando las funciones en la propia base de datos ya que MongoDB ofrece una colección llamada *system.js* que permite el almacenamiento de funciones JavaScript [21]. Pero al igual que ocurría con la otra opción, tiene algunos inconvenientes:

  - Al ser una colección en la que se almacena el código, cada vez que se necesitan hacer cambios en el mismo es necesario actualizar la entrada donde se ha almacenado, es decir, realizar una sentencia de Update. Esto sumado a que la llamada a la función también se realiza mediante el código de consulta de Mongo hace muy tedioso el desarrollo de esta forma.
  - En la propia página de la documentación de MongoDB advierten que no es buena práctica almacenar mucha lógica de la aplicación en esta colección, que está destinada simplemente a almacenar funciones sin mucha carga de ejecución pues que el hecho de estar almacenada en el lado del servidor, disminuye el rendimiento de la misma.

Por tanto, y como observamos en los puntos anteriores, ambos modos de integrar el código JavaScript directamente con MongoDB presentan grandes inconvenientes que hacen que se intente evitar la elección de esta alternativa. Además es necesario mencionar que la ejecución de código JavaScript en el intérprete de MongoDB está limitada simplemente a las operaciones nativas que permite dicho lenguaje de programación, no se pueden añadir módulos ni ningún tipo de librería externa al mismo.

3.5.2.2. Node.js + MongoDB

Esta alternativa hace uso del framework Node.js para integrar las funcionalidades del sistema. Node.js es un intérprete de JavaScript que se ejecuta en el lado del servidor y que, unida a MongoDB, en los últimos años ha ido creciendo considerablemente su uso y volviéndose más popular a la hora de desarrollar aplicaciones de forma rápida y sencilla.

No obstante, no es esto lo que se buscaba a la hora de realizar esta propuesta de diseño, ya que crear una aplicación completa a partir del alcance de este proyecto se torna inabarcable para el tiempo del que se dispone. Se ha escogido esta alternativa debido a

la flexibilidad que presenta el framework Node.js, y que a diferencia del interprete de la consola de MongoDB, este si permite la adición de módulos y librerías externas que puedan resultar de ayuda a la hora de realizar el desarrollo de una forma más simple y rápida.

Además, también cabe destacar de esta propuesta que permite jerarquizar el código. A pesar de que el lenguaje de programación JavaScript no está orientado a objetos, es decir, no podemos organizar el código con clases tal y como sí permiten otros lenguajes como Java, si que permite cierta estructura por lo que lo convierte en un entorno óptimo para el desarrollo junto con la característica más importante a nuestro criterio: la posibilidad de generar varios archivos de código JavaScript e importar dicho código en otros scripts distintos. Algo que no se permitía realizar en la propuesta anterior debido a las restricciones de integración con MongoDB que presentaba.

En cuanto a la conexión con MongoDB, ahora esto resulta un inconveniente, puesto que al tratarse de un framework externo al software no disponemos de una conexión intrínseca con el desarrollo como ocurría en la propuesta anterior. No obstante, como se ha dicho antes, Node.js es un framework muy popular, y fruto de esa popularidad posee un ecosistema de paquetes, npm, que se trata del catalogo de librerías de código abierto más grande del mundo. No es extraño encontrar entonces múltiples de estas librerías que se encargan de realizar una conexión a la base de datos de MongoDB de forma totalmente transparente, así como utilizar las funcionalidades básicas del software de base de datos (inserciones, borrados, consultas...). De entre todos los que se ofertan en npm se ha seleccionado para esta propuesta, la utilización del módulo oficial ofrecido por MongoDB.

Además, dada esta característica de Node.js de poseer tal ecosistema de librerías, se puede aprovechar la situación y orientar el desarrollo de las funcionalidades a la creación de un nuevo módulo que pase a formar parte de ese gran conjunto de librerías de código abierto. De esta manera este trabajo se sitúa como un desarrollo que propone implementar las funcionalidades definidas en el alcance del trabajo en forma de API. Se cuenta además con la ventaja de que no es necesario que el usuario instale todas las librerías que se utilizan (en este caso, para realizar la conexión con MongoDB), puesto que el entorno de creación de módulos de Node.js permite la adición de dependencias (es decir, otros módulos que se han usado a la hora de desarrollar el código) para que se instalen automáticamente cuando se instala la librería en cuestión.

Si se realiza de esta manera el desarrollo, a diferencia de lo que ocurría con la alternativa del apartado anterior, permite una instalación sencilla al alcance de cualquier usuario, ya que no depende de la instalación de MongoDB sino de tener el entorno de Node.js instalado. Obviamente si se pretende que funcione el sistema final una vez desarrollado si que necesitaremos disponer de alguna instancia de MongoDB, bien sea en un servidor o en una máquina local.

3.5.2.3. Node.js + MongoDB Compass

La última de las propuestas consiste en utilizar la aplicación provista por el equipo de MongoDB, Compass[1] en conjunto con Node.js para incorporar las funcionalidades que se pretenden desarrollar, en el escenario de una interfaz gráfica.

Esta herramienta permite visualizar los datos y realizar las operaciones que permite Mongo de una forma más amigable para el usuario, puesto que este trabaja en un entorno al que ya está acostumbrado mediante la interacción con ventanas, botones, etc. En definitiva, una interfaz gráfica completa que permite realizar las mismas acciones que la consola de MongoDB pero de forma más sencilla dado que no hay que poseer tanto conocimiento técnico (solo el pertinente al lenguaje de consultas con el que trabaja MongoDB). Cabe destacar que esto no supone ningún problema para el desarrollador, que goza de los conocimientos técnicos necesarios, pero la razón de implementarlo de esta manera, es la de poder llegar a un público más amplio gracias al uso de una interfaz que permita realizar las operaciones que debe permitir nuestro sistema.

Por tanto, para esta propuesta se establece utilizar dicha herramienta a la hora de implementar las funcionalidades. La razón de que hayamos mencionado que esta solución es prácticamente igual que la anterior no es otra que la forma de implementación que tiene: también utiliza el framework de Node.js para el desarrollo de nuevas funcionalidades de la aplicación (entre otros) tal y como se indica en el la documentación sobre la adición de nuevos módulos a la herramienta Compass [22]. Por tanto, la implementación jerarquizada en código tal y como se concibe en la alternativa anterior permanece intacta, simplemente se añade la tecnología necesaria para desarrollar la interfaz.

Dicha tecnología no es otra que React.js [2]. Un framework también de JavaScript que permite la creación de interfaces. No obstante, aquí encontramos el primer inconveniente, y es que aunque el equipo desarrollador conoce JavaScript, no posee los conocimientos del funcionamiento de React.js. Además a esto debemos añadirle que a la realización de la lógica del código hay que sumar el desarrollo de una interfaz, lo cual produce que el tiempo empleado en el desarrollo sea mayor.

3.5.2.4. Elección final de la tecnología

Una vez propuestas estas tres alternativas, se ha elegido la que más se adapta a este proyecto y reúne las mejores características para que resulte en una solución final de calidad.

En cuanto a la primera, cuando se ha realizado la exposición de la alternativa y expuesto sus puntos fuertes y débiles, se ha comprobado que tenía más de estos últimos que de los primeros. por lo que se ha descartado rápidamente al no resultar en una solución óptima que cumpla los requisitos propuestos.

[1] `https://www.mongodb.com/products/compass`
[2] `https://reactjs.org/`

La segunda opción mejora los aspectos que no resultaban convincentes en la primera, y presenta un entorno de desarrollo óptimo en el que poder construir la solución final de la manera más cómoda posible a nuestro criterio. Además de la ventaja de poder desplegar dicha solución como un modelo de abstracción que permita el uso de las funcionalidades de la solución a otros usuarios.

Finalmente, la tercera opción, se trata de una alternativa que en gran parte engloba la segunda, con el añadido de crear una interfaz gráfica para el programa MongoDB Compass. Esto puede resultar muy ambicioso y, aunque sería idóneo realizar una interfaz para disponer de un sistema final utilizable (al contrario que la API que se trata de una alternativa que actúa como puente para lograr un objetivo), excede las metas que se propusieron en este trabajo al tratarse del desarrollo de una aplicación completa.

Por tanto, bajo estas premisas, se ha escogido la segunda alternativa: MongoDB + Node.js al resultar ser la que mejores características reúne para la realización de este proyecto.

3.6. Especificaciones del sistema

En esta sección se establecen las especificaciones de software y hardware a las que se acoge este proyecto en forma de requisitos. Además se establecen las características físicas del entorno de pruebas en el que se va a desarrollar este proyecto.

3.6.1. Establecimiento de requisitos

A continuación se presenta la extracción de requisitos software que se ha realizado una vez que se dispone de la información tanto funcional (qué se va a hacer) como técnica (qué herramientas se van a utilizar). Con esto se persigue el objetivo de tener una visión más clara de cómo realizar la solución final.

No obstante se destaca que pese a que reciben el calificativo de "Requisito de Software", dada la naturaleza investigadora del trabajo resulta imposible establecer unos requisitos tan concisos como cabría esperar. Esto sucede debido al hecho que hasta que no se inicie la etapa de implementación y se tenga conocimiento de los resultados que el sistema va arrojando, no es posible conocer, por ejemplo, que algoritmos de técnicas difusas son mejores frente a otros. Es por esto que en ese caso concreto, se establece un requisito muy general que permite el cumplimiento de la finalidad de la solución, pero no se trata de un requisito estricto en el que se determina exactamente qué algoritmo se implementa.

Dicho esto, para una buena organización de los requisitos, se ha establecido un formato de tabla en el que se recogen todos ellos. El formato que se ha seguido es el siguiente:

RS-XX	
Título :	*Título del requisito*
Descripción :	*Descripción del requisito*
Procedencia :	*Relación con los requisitos funcionales*
Prioridad :	*Alta-Baja*

Tabla 3.2. Plantilla de requisitos

Dónde las celdas que lo componen se refieren a:

- **RS-XX**: Indica que se trata de un Requisito de Software (RS) con XX como la numeración de cada requisito. Ambos valores forman el identificador de un requisito.
- **Título**: Descripción breve del requisito.
- **Descripción**: Explicación detallada del requisito.
- **Relación**: Indica la relación que hay entre el requisito de software y el funcional. Todos los requisitos funcionales deben estar reflejados en los requisitos de software.

- **Prioridad**: Indica la importancia de que el requisito se encuentre incluido en el sistema. Un valor de "Alta" indica que el sistema es incapaz de funcionar sin el requisito, mientras que un valor de "Baja" indica que el sistema puede ser perfectamente funcional sin el mismo.

A continuación se exponen todos los requisitos identificados siguiendo el modelo de tabla propuesto.

RS-01	
Título :	Software NoSQL
Descripción :	El Software NoSQL que debe utilizarse para la realización del sistema es MongoDB en la versión 3.6.5 Community Server
Procedencia :	RF-01
Prioridad :	Alta

Tabla 3.3. RS-01: Software NoSQL

RS-02	
Título :	Código del Sistema
Descripción :	El Código del sistema deberá desarrollarse en JavaScript
Procedencia :	RF-01
Prioridad :	Alta

Tabla 3.4. RS-02: Código del sistema

RS-03	
Título :	Framework Node.js
Descripción :	El sistema deberá implementarse haciendo uso del framework Node.js en su versión 8.11.2 LTS
Procedencia :	RF-01 RF-02
Prioridad :	Alta

Tabla 3.5. RS-03: Framework Node.js

RS-04	
Título :	Paquete npm
Descripción :	El sistema debe codificarse siguiendo el modelo de paquete npm
Procedencia :	RF-02 RF-03
Prioridad :	Alta

Tabla 3.6. RS-04: Paquete npm

RS-05	
Título :	Conexión MongoDB
Descripción :	La conexión con MongoDB se realizará a través del driver oficial para MongoDB[3] de npm
Procedencia :	RF-01 RF-02 RF-03
Prioridad :	Alta

Tabla 3.7. RS-05: Conexión MongoDB

RS-06	
Título :	Compatibilidad con base de datos
Descripción :	El sistema deberá ser funcional con cualquier base de datos de MongoDB
Procedencia :	RF-01 RF-03
Prioridad :	Alta

Tabla 3.8. RS-06: Compatibilidad con base de datos

RS-07	
Título :	Compatibilidad con colección
Descripción :	El sistema deberá ser funcional con cualquier colección almacenada en una base de datos de MongoDB
Procedencia :	RF-01 RF-03
Prioridad :	Alta

Tabla 3.9. RS-07: Compatibilidad con colección

[3] `https://www.npmjs.com/package/mongodb`

RS-08	
Título :	Comparación difusa
Descripción :	El sistema deberá implementar la comparación difusa de documentos
Procedencia :	RF-04
Prioridad :	Alta

Tabla 3.10. RS-08: Comparación difusa

RS-09	
Título :	Compatibilidad con tipos de dato
Descripción :	La comparación de documentos se realizará independientemente de los tipos de dato que compongan el documento
Procedencia :	RF-04
Prioridad :	Alta

Tabla 3.11. RS-09: Compatibilidad con tipos de dato

RS-10	
Título :	Compatibilidad con estructura
Descripción :	La comparación de documentos se realizará independientemente de la estructura que tenga el documento
Procedencia :	RF-04
Prioridad :	Alta

Tabla 3.12. RS-10: Compatibilidad con estructura

RS-11	
Título :	Similitud entre cadenas
Descripción :	Para determinar la similitud de cadenas se implementarán los algoritmos de aproximación de cadenas Jaccard, Kondrak y Levenshtein
Procedencia :	RF-04 RF-05
Prioridad :	Alta

Tabla 3.13. RS-11: Similitud entre cadenas

RS-12	
Título :	Similitud entre números
Descripción :	Para determinar la similitud entre números se implementarán algoritmos de pertenencia difusa
Procedencia :	RF-04 RF-05
Prioridad :	Alta

Tabla 3.14. RS-12: Similitud entre números

RS-13	
Título :	Similitud entre fechas
Descripción :	Para determinar la similitud entre fechas se implementarán algoritmos de pertenencia difusa
Procedencia :	RF-04 RF-05
Prioridad :	Alta

Tabla 3.15. RS-13: Similitud entre números

RS-14	
Título :	Similitud entre arrays
Descripción :	Para determinar la similitud entre arrays se utilizará el algoritmo de Jaccard (pertenencia difusa a un conjunto)
Procedencia :	RF-04 RF-05
Prioridad :	Alta

Tabla 3.16. RS-14: Similitud entre arrays

RS-15	
Título :	Similitud entre documentos anidados
Descripción :	Para determinar la similitud entre documentos anidados se aplicará el algoritmo de comparación de documentos de forma recursiva
Procedencia :	RF-04 RF-05
Prioridad :	Alta

Tabla 3.17. RS-15: Similitud entre documentos anidados

RS-16	
Título :	Inserción difusa
Descripción :	El sistema comprobará si existe un documento similar insertando solamente en caso de no existencia
Procedencia :	RF-03 RF-04 RF-06
Prioridad :	Alta

Tabla 3.18. RS-16: Inserción difusa

RS-17	
Título :	Borrado difuso
Descripción :	El sistema analizará la base de datos documento a documento, eliminando los documentos que sean similares al analizado
Procedencia :	RF-03 RF-04 RF-07
Prioridad :	Alta

Tabla 3.19. RS-17: Borrado difuso

RS-18	
Título :	Recuperación difusa
Descripción :	El sistema retornará los documentos similares al indicado en caso de existir
Procedencia :	RF-03 RF-04 RF-08
Prioridad :	Alta

Tabla 3.20. RS-18: Recuperación difusa

RS-19	
Título :	Umbral de similitud
Descripción :	Se podrá especificar el valor mínimo de similitud que debe tomar el sistema para determinar cuando un documento es similar o no
Procedencia :	RF-03 RF-04
Prioridad :	Alta

Tabla 3.21. RS-19: Umbral de similitud

RS-20	
Título :	Algoritmo comparador optimizado
Descripción :	El sistema dispondrá de un algoritmo comparador que tomará el documento como un string sin diferenciar entre tipos de dato
Procedencia :	RF-03 RF-04
Prioridad :	Baja

Tabla 3.22. RS-20:Inserción difusa

RS-21	
Título :	Elección de algoritmo
Descripción :	Para el algoritmo comparador optimizado, el sistema realizará la comparación con aproximación de cadenas, dejando el algoritmo utilizado a elección del usuario.
Procedencia :	RF-03 RF-04
Prioridad :	Baja

Tabla 3.23. RS-21: Elección de algoritmo

RS-22	
Título :	Funcionalidad asíncrona
Descripción :	El sistema deberá realizar la ejecución de las funciones de forma asíncrona (característica de Node.js)
Procedencia :	RF-03 RF-04
Prioridad :	Alta

Tabla 3.24. RS-22: Funcionalidad asíncrona

RS-23	
Título :	Visualización de ejecución
Descripción :	Durante la ejecución asíncrona de las tres funcionalidades clave se muestra una barra de carga
Procedencia :	RF-03 RF-04
Prioridad :	Baja

Tabla 3.25. RS-23: Funcionalidad asíncrona

A continuación, se puede ver la matriz de trazabilidad entre los requisitos de software identificados y los funcionales, de esta manera se asegura que todos los requisitos

funcionales del sistema que se pretende desarrollar están cubiertos por los requisitos de software.

RS \RF	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08
RS-01	X							
RS-02	X							
RS-03	X	X						
RS-04		X	X					
RS-05	X	X	X					
RS-06	X		X					
RS-07	X		X					
RS-08				X				
RS-09				X				
RS-10				X				
RS-11				X	X			
RS-12				X	X			
RS-13				X	X			
RS-14				X	X			
RS-15				X	X			
RS-16			X	X		X		
RS-17			X	X			X	
RS-18			X	X				X
RS-19			X	X				
RS-20			X	X				
RS-21			X	X				
RS-22			X	X				
RS-23			X	X				

Tabla 3.26. Matriz de trazabilidad: Requisitos Funcionales y Requisitos Software

3.6.2. Especificación del entorno de pruebas

Una vez se ha alcanzado este punto, ya conocemos con precisión la tecnología se la que se va a hacer uso para el desarrollo del proyecto así como las funciones que este debe realizar. Por tanto es necesario que se contemple también las especificaciones del entorno de pruebas; es decir, bajo que condiciones se van a efectuar las pruebas una vez que se haya desarrollado el sistema en cuestión.

Para la realización de las pruebas se ha utilizado un equipo cuyas características son las siguientes:

- Sistema Operativo Windows 10 de 64 bits.

- 2GB de RAM.
- Procesador Intel® Cherry Trail Quad-Core Z8300 1.84 GHz.
- Almacenamiento 32GB SSD.

Dada la limitada capacidad de cómputo del equipo se ha utilizado una base de datos pequeña puesto que si no la carga de trabajo para el equipo iba a resultar muy grande. Además de la gran cantidad de tiempo que conllevaría ejecutar las operaciones difusas para las pruebas con una base de datos de grandes dimensiones (Puesto que cuanta más información tengamos, más tiempo se tardará en procesar).

Además se necesita una base de datos que contenga estructuras variables y haga uso de todos los tipos de dato posibles para probar correctamente el que nuestro sistema cumple los objetivos establecidos. Funciona para todos los tipos de datos y todos los casos posibles.

Por estas razones se ha utilizado una base de datos de ejemplo propuesta en la página we de MongoDB [4] compuesta por una colección de 10 documentos. Con esta base de datos se permitirá el desarrollo completo de las pruebas en el equipo con las especificaciones que se han mencionado.

Además, para la realización de las pruebas se debe tener en cuenta que el peso del proyecto recae en la operación de comparación; es decir, establecer un valor de similitud y gracias al mismo, implementar las operaciones difusas. Por ello es importante destacar que al conjunto predefinido de datos que se utilizará se añade el siguiente documento:

[4] `https://docs.mongodb.com/manual/reference/bios-example-collection/`

```
{
    "_id" : 11,
    "name" : {
        "first" : "Dennisse",
        "last" : "Ritchson"
    },
    "birth" : Date("1941-09-11"),
    "death" : Date("2011-10-14"),
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1984,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1995,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2009,
            "by" : "The Japan Prize Foundation"
        }
    ]
}
```

Fig. 3.7. Documento introducido para las pruebas

En el cual se han modificado solamente algunos campos de otro documento que pertenece a la base de datos utilizada. El documento almacenado en la base de datos similar al que se ha expuesto es el siguiente:

```
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00Z"),
    "death" : ISODate("2011-10-12T04:00:00Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1998,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2011,
            "by" : "The Japan Prize Foundation"
        }
    ]
}
```

Fig. 3.8. Documento alamacenado en la BBDD similar al introducido

Como podemos observar, las diferencias (a parte del _id, pero este no se tiene en cuenta a la hora de realizar la similitud) son las siguientes:

- Los dos atributos que componen el objeto nombre son muy similares. En el caso del campo “First” tenemos Dennisse y Dennis, mientras que en el campo “Last” tenemos Ritchson y Ritchie.
- En el caso de los campos “birth” y “date”, el documento nuevo insertado tiene unas fechas dos días posteriores al documento de la base de datos original.
- En el caso del atributo “contribs”, el documento nuevo no lo tiene contenido.
- Finalmente, en el caso del atributo “awards”, un array de objetos, el documento insertado tiene los mismos premios que el almacenado en la base de datos con la

diferencia de los años, que en el caso del que se ha insertado son 1984, 1995 y 2009 respectivamente.

Para referirnos a estos dos documentos, llamaremos al insertado documento de “Dennisse Ritchson” y al almacenado previamente documento de “Dennis Ritchie”.

3.7. Establecimiento de las pruebas

Una vez se han definido los requisitos que la solución tendrá que implementar y el entorno de pruebas en el que se va a operar, se debe proceder a la siguiente tarea del análisis: la definición de un plan de pruebas. El objetivo que se busca conseguir no es otro que dar una guía para la realización de pruebas de nuestro sistema, que se realizará más adelante en el capítulo *"Descripción de resultados"*, y verificar que este cumple con los requisitos propuestos en los puntos anteriores.

Es importante destacar que existen multitud de pruebas que pueden realizarse sobre el software, pero en función de la solución adoptada y la situación concreta de cada sistema, es posible que determinadas pruebas sean más importantes que otras. Un ejemplo claro de ello son las pruebas de implantación, a las que no daremos mucha importancia debido a que realizar la instalación de nuestro sistema a través del entorno de paquetes de npm resultará muy sencillo, y por tanto no describiremos este nivel de pruebas. O las pruebas de carga, en las que se mide la capacidad de respuesta del sistema. Puesto que nuestra solución no contempla el acceso concurrente de múltiples usuarios tampoco se contempla este nivel de pruebas.

De esta forma, para este caso, se han distinguido entre tres niveles de pruebas [23] que se pueden aplicar a este sistema. Estas son:

- Pruebas unitarias: Estas pruebas son las que se realizan sobre una unidad de código. En otras palabras, con ellas se comprueba el correcto funcionamiento del código desarrollado y que este no produce errores durante la ejecución. Son las primeras que se llevan a cabo en las especificaciones de pruebas.

  En el caso que nos ocupa, estas pruebas se realizarán sobre el funcionamiento de los algoritmos implementados, es decir, se usarán en la implementación en código de los algoritmos de aproximación de cadenas y pertenencia difusa, para comprobar que funcionan correctamente.

- Pruebas de Integración: Estas pruebas se realizan para detectar fallos en la interacción de las pequeñas porciones de código.

  En el caso que nos ocupa, se trata de un desarrollo que se va efectuando de forma incremental puesto que primero realizamos la implementación de los algoritmos, posteriormente del comparador de documentos y finalmente de las funciones difusas sobre MongoDB. Por esto, se realizarán las pruebas de integración sobre la interacción entre la implementación de los algoritmos y el comparador.

- Pruebas de Sistema: Estas pruebas son las últimas que se aplican, y verifican que los requisitos expuestos en el análisis del alcance del trabajo se cumplen en el código, es decir, que el sistema realiza todo aquello que se pretendía en un primer momento de forma correcta.

En este caso, se realizarán estas pruebas sobre el sistema final, de manera que se podrá comprobar que la funcionalidad final del sistema es correcta. En otras palabras, se comprueba que realiza la inserción difusa, el borrado difuso y la recuperación difusa.

Por tanto, y una vez introducida la ruta que se va a seguir en el plan de pruebas, gracias a los requisitos, el alcance del trabajo que va a tener este proyecto y el entorno de pruebas que se ha establecido en puntos anteriores, se identifican las pruebas que se van a realizar.

En los siguientes apartados, organizados según los tipos de prueba, se describe en forma de tablas las pruebas que se han realizado sobre el sistema. La tabla que se ha usado para representar cada prueba ha sido la siguiente:

P/(U-I-S)-XX	
Título :	*Título de la prueba*
Descripción :	*Descripción de la prueba*
Resultado esperado :	*Resultado esperado*
Procedencia :	*Requisito Software*

Tabla 3.27. P/(U-I-S)-XX: Plantilla de pruebas

Dónde los campos de las tablas indican lo siguiente:

- **P/(U-I-S)-XX** : Identifica si se trata de una prueba unitaria cuando adquiere el valor PU, una prueba de integración con el valor PI o una prueba del sistema con el valor PS. XX indica la numeración de cada tipo de prueba. Ambos valores en conjunto forman el identificador de la prueba.
- **Título** : Título que describe brevemente la prueba.
- **Descripción** : Explicación detallada de la prueba.
- **Resultado esperado** : Resultado que se espera de la prueba.
- **Procedencia** : Indica la relación que existe entre la prueba y los requisitos de software a partir de los cuales se ha establecido la misma.

3.7.1. Pruebas Unitarias

PU-01	
Título :	Algoritmo de Levenshtein: resultado positivo
Descripción :	Se introducen dos cadenas de caracteres; “Kitten” y “Sitten” para evaluar el resultado de la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea muy cercano a 1 puesto que ambas palabras difieren en una sola letra.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.28. PU-01: Algoritmo de Levenshtein: resultado positivo

PU-02	
Título :	Algoritmo de Levenshtein: resultado negativo
Descripción :	Se introducen dos cadenas de caracteres; “Kitten” y “Dogmeat” para evaluar el resultado de la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea muy cercano a 0 puesto que las palabras propuestas no poseen apenas coincidencias.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.29. PU-02: Algoritmo de Levenshtein: resultado negativo

PU-03	
Título :	Algoritmo de Jaccard: resultado positivo
Descripción :	Se introduce “El perro de Laura” y “El perro de Lars” para evaluar el resultado de la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 1.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.30. PU-03: Algoritmo de Jaccard: resultado positivo

PU-04	
Título :	Algoritmo de Jaccard: resultado negativo
Descripción :	Se introducen las cadenas “El perro de Laura” y “La gata de David” para la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 0 puesto que las cadenas de caracteres no tienen apenas ngrams en común.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.31. PU-04: Algoritmo de Jaccard: resultado negativo

PU-05	
Título :	Algoritmo de Kondrak: resultado positivo
Descripción :	Se introducen las cadenas “El perro de Laura” y “El perro de Lars’ para la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 1 puesto que las cadenas de caracteres son a priori muy similares.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.32. PU-05: Algoritmo de Kondrak: resultado positivo

PU-06	
Título :	Algoritmo de kondrak: resultado negativo
Descripción :	Se introducen las cadenas “El perro de Laura” y “La gata de David” para la similitud.
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 0 puesto que las cadenas de caracteres no son muy similares.
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.33. PU-06: Algoritmo de kondrak: resultado negativo

PU-07	
Título :	Algoritmo de Gauss: resultado positivo
Descripción :	Se compara el número 10 con 12,5
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 1 puesto que con el valor de anchura de la campana indicado los números son muy cercanos entre sí.
Procedencia :	RS-02 RS-03 RS-12 RS-13

Tabla 3.34. PU-04: Algoritmo de Gauss: resultado positivo

PU-08	
Título :	Algoritmo de Gauss: resultado negativo
Descripción :	Se compara el número 10 con 12,5
Resultado esperado :	Se espera que el resultado de similitud sea cercano a 0 puesto que para el valor de anchura indicado los números no resultan similares al indicado.
Procedencia :	RS-02 RS-03 RS-12 RS-13

Tabla 3.35. PU-08: Algoritmo de Gauss: resultado negativo

PU-09	
Título :	Fragmentar cadena en ngrams
Descripción :	Se pretende fragmentar la cadena .El perro de Larsçon ngrams de tamaño 3.
Resultado esperado :	Se espera que se fragmente la cadena de la manera siguiente: [elp, lpe, per, err, rro, rod, ode, del, ela, lar, ars].
Procedencia :	RS-02 RS-03 RS-11

Tabla 3.36. PU-09: Fragmentar cadena de ngrams

PU-10	
Título :	Mostrar Spinner en la pantalla
Descripción :	Se comprobará si la barra de carga se muestra en la consola
Resultado esperado :	La aparición en la consola de una barra de carga que se muestre durante el tiempo de ejecución
Procedencia :	RS-02 RS-03 RS-22 RS-23

Tabla 3.37. PU-10: Mostrar Spinner en la pantalla

PU-11	
Título :	Algoritmo de similitud para arrays
Descripción :	Se compara el array [“Laura”, 12.3, “El perro es un gran danés”] con [“Lars”,12.1, “El perro es un gran amigo”]
Resultado esperado :	Se espera que el resultado de similitud sea 0,5 puesto que para el primer elemento del array no se va a considerar como elemento similar mientras que para los otros dos si, por tanto la cardinalidad de la intersección va a ser 2 mientras que la unión va a ser 4. Esto da un valor de 0.5
Procedencia :	RS-02 RS-03 RS-14

Tabla 3.38. PU-11: Algoritmo de similitud para arrays

3.7.2. Pruebas de integración

PI-01	
Título :	Comparador de documentos: Comparación precisa
Descripción :	Se compara el documento que representa a “Dennis Ritchie” con “Dennisse Ritchson”
Resultado esperado :	Se espera que el resultado de similitud sea más cercano a uno que a cero debido a que los documentos que se van a comparar tienen bastante puntos en común
Procedencia :	RS-02 RS-03 RS-08 RS-09 RS-10 RS-15

Tabla 3.39. PI-01: Comparador de documentos: Comparación precisa

PI-02	
Título :	Comparador de documentos: Comparación serializada
Descripción :	Se compara el documento que representa a “Dennis Ritchie” con “Dennisse Ritchson”
Resultado esperado :	Se espera que el resultado de similitud sea más cercano a uno que a cero debido a que los documentos que se van a comparar tienen bastante puntos en común
Procedencia :	RS-02 RS-03 RS-20 RS-21

Tabla 3.40. PI-02: Comparador de documentos: Comparación serializada

3.7.3. Pruebas de Sistema

PS-01	
Título :	Inserción difusa fallida
Descripción :	Se intentará insertar el documento utilizado para la comparación en el apartado anterior. Se establece un umbral del 0.7
Resultado esperado :	Se espera que no se inserte el documento debido a que es muy similar al documento “Dennis Ritchie” almacenado.
Procedencia :	RS-01 RS-02 RS-03 RS-04 RS-05 RS-06 RS-07 RS-16 RS-19 RS-22 RS-23

Tabla 3.41. PS-01: Inserción difusa fallida

PS-02	
Título :	Inserción difusa correcta
Descripción :	Se intentará insertar el documento utilizado para la comparación en el apartado anterior. Se establece un umbral del 0.8
Resultado esperado :	Se espera que se inserte el documento debido a que, a pesar de ser similar al documento “Dennis Ritchie” que ya se encuentra almacenado, no alcanza el umbral de 0,8 y tampoco supera ese umbral de similitud para el resto de documentos almacenados.
Procedencia :	RS-01 RS-02 RS-03 RS-04 RS-05 RS-06 RS-07 RS-16 RS-19 RS-22 RS-22

Tabla 3.42. PS-02: Inserción difusa correcta

PS-03	
Título :	Inserción difusa múltiple
Descripción :	Se intentará insertar el documento utilizado para la comparación en la sección anterior y uno nuevo al que solamente se le indica el campo del nombre: “Damian Reitch”. Se establece un umbral del 0.7
Resultado esperado :	Se espera que no se inserte el primer documento ya que supera el umbral de similitud para el documento “Dennis Ritchie” que ya se encuentra almacenado. El segundo documento se insertará al no ser similar a ninguno que se encuentre almacenado.
Procedencia :	RS-01 RS-02 RS-03 RS-04 RS-05 RS-06 RS-07 RS-16 RS-19 RS-22 RS-22

Tabla 3.43. PS-03: Inserción difusa múltiple

PS-04	
Título :	Limpieza de la colección
Descripción :	Se limpia la colección previa inserción del documento que se utilizaba en la sección anterior. Se establece un umbral del 0.7
Resultado esperado :	Se espera que se elimine el documento insertado debido a que es muy similar al documento de “Dennis Ritchie”.
Procedencia :	RS-01 RS-02 RS-03 RS-04 RS-05 RS-06 RS-07 RS-17 RS-19 RS-22 RS-22

Tabla 3.44. PS-04: Limpieza de la colección

PS-05	
Título :	Recuperación difusa
Descripción :	Se recupera el documento de “Dennis Ritchie” indicando el documento utilizado en la sección anterior para realizar las comparaciones. Se establece un umbral del 0.7
Resultado esperado :	Se espera que se recupere el documento “Dennis Ritchie” al superar el umbral de similitud de 0,7 al compararlo con el de la sección anterior.
Procedencia :	RS-01 RS-02 RS-03 RS-04 RS-05 RS-06 RS-07 RS-18 RS-19 RS-22 RS-22

Tabla 3.45. PS-05: Recuperación difusa

En la siguiente página se muestra la matriz de trazabilidad para comprobar que todas las pruebas realizadas cubren los requisitos especificados anteriormente.

RS \P	PU-01	PU-02	PU-03	PU-04	PU-05	PU-06	PU-07	PU-08	PU-09	PU-10	PU-11	PI-01	PI-02	PS-01	PS-02	PS-03	PS-04	PS-05
RS-01														X	X	X	X	X
RS-02	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RS-03	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RS-04														X	X	X	X	X
RS-05														X	X	X	X	X
RS-06														X	X	X	X	X
RS-07														X	X	X	X	X
RS-08												X						
RS-09												X						
RS-10												X						
RS-11	X	X	X	X	X	X			X									
RS-12							X	X										
RS-13							X	X										
RS-14											X							
RS-15												X						
RS-16														X	X	X		
RS-17																	X	
RS-18																		X
RS-19														X	X	X	X	X
RS-20													X					
RS-21													X					
RS-22										X				X	X	X	X	X
RS-23										X				X	X	X	X	X

Tabla 3.46. Matriz de trazabilidad: Requisitos Software y Pruebas

3.8. Definición del ciclo de vida

Tras la realización de la especificación de requisitos, se debe establecer que método de trabajo va a seguirse durante todo el ciclo de vida del proyecto.En primer lugar se ordena la realización de las tareas. Como se ha mencionado con anterioridad, para llevar a cabo la solución final se va a realizar un módulo que sea capaz de comparar documentos completos, ya que sin el mismo, resulta imposible realizar la implementación de las operaciones difusas especificadas. A su vez, este módulo necesita de la implementación de algoritmos, tanto pertenecientes a aproximación de cadenas para los tipos de dato String y Arrays, como de pertenencia difusa para los tipos de dato númericos y fechas (tal y como se detalla en los requisitos establecidos en el apartado anterior).

En resumen, para implementar las funciones difusas sobre MongoDB es necesaria la implementación de un módulo que compare documentos, y a su vez para la implementación de dicho módulo se necesita de la implementación de algoritmos de técnicas difusas. Además, cabe destacar que debido a la solución tecnológica que se ha escogido (Node.js + MongoDB) se tiene la posibilidad de estructurar el código necesario para alcanzar esos objetivos de forma independiente, en otras palabras, se podrán desarrollar módulos separados que interaccionen entre ellos para lograr la finalidad propuesta. Es por estas razones que el trabajo que debemos desarrollar se ha separado en tres fases:

- Fase 1 : Implementación de los algoritmos difusos en el lenguaje JavaScript.
- Fase 2 : Implementación del algoritmo comparador de documentos haciendo uso de los algoritmos difusos.
- Fase 3 : Implementación de las funciones de inserción, borrado y recuperación difusa haciendo uso del comparador de documentos.

Por tanto será necesario escoger una metodología que permita la división en estas tres fases a la hora de realizar la implementación.

3.9. Metodología a implementar

Una vez determinada la que se ha división de nuestro trabajo, se procede a escoger una metodología para llevarla a cabo de forma exitosa. De esta manera se capacita al equipo para marcarse objetivos a lo largo del ciclo de vida del proyecto e ir cumpliéndolos a medida que avanza el tiempo.

Debido a que el desarrollo de este trabajo se ha divido en las tres fases mencionadas, de entre las metodologías de software mas comunes para la realización de este proyecto se barajó utilizar las siguientes: secuencial, iterativa, incremental y espiral, ya que todas ellas poseen la característica de organizar el trabajo a realizar a través de ciclos o fases [24].

La metodología iterativa se descartó debido a que una de sus características más fuertes es que los requisitos no se encuentran bien detallados al comienzo por lo que necesita de un primer prototipo de la aplicación [25]. Se podría pensar que esta situación se asemeja a este proyecto ya que los requisitos anteriormente definidos se pueden cumplir con ligeras variaciones y ser susceptibles a cambios, pero la realización de un prototipo es totalmente incompatible con este proyecto.

En cuanto a la metodología incremental, esta se centra más en el aspecto de "requisitos cambiantes" por lo que se puede realizar un desarrollo en función de las necesidades que se presenten a lo largo del ciclo de vida del proyecto [25]. Si bien esto se asemeja a la situación que nos ocupa, nos percatamos de que cada iteración en este modelo constituye una versión ya completa de la aplicación. Esta situación no aplica a este caso ya que al dividir cada iteración en las fases descritas, por si solas no dan lugar a la aplicación completa.

La metodología secuencial (más conocida como "metodología en cascada") se descartó rápidamente debido a que necesita disponer de una definición de requisitos clara y completa. Este tipo de metodología se basa principalmente en la utilización de una amplia documentación escrita fruto de realizar un análisis previo a la implementación. En este caso esto no convenía, ya que, pese a que el análisis se esta detallando en el capítulo que nos ocupa, no conocemos a ciencia cierta la profundidad de este trabajo (variando en función de los objetivos y el resultado obtenido en las fases en las que se ha dividido el proyecto), por lo que resulta imposible generar toda la documentación completa antes de llevar a cabo la implementación [24].

Por tanto, y una vez descartadas las metodologías anteriores, la metodología en espiral ha sido la seleccionada ya que esta permite fragmentar el trabajo del proyecto en fases. De esta manera se puede realizar el desarrollo de la aplicación de forma secuencial e iterativa pero separando el proceso por cada fase en: diseño y análisis, desarrollo, pruebas y documentación. Además, al contrario que los modelos anteriormente descritos, no necesita la presentación de un prototipo al comienzo del desarrollo ni la generación completa de la documentación y especificación de requisitos inamovibles [25].

El modelo en espiral fue propuesto en 1988 por Barry W. Boehm [26]. Como ya se ha descrito anteriormente, el principal objetivo de este modelo es el de fragmentar el proyecto en fases de manera que cada ciclo de la espiral (es decir, cada vuelta). De esta forma cada ciclo corresponde a cada una de las tres fases que componen nuestro proyecto: El estudio e implementación de los algoritmos difusos, la ideación de un comparador de documentos difuso y finalmente la integración con las funciones ofrecidas por MongoDB.

Cada ciclo de la espiral a su vez, se divide en cuatro sectores en los que siempre se realizan las mismas acciones para cada fase:

- Definir Objetivos : En esta fase del proyecto se definen los objetivos que se pretenden alcanzar en dicha fase. Se analiza la situación además de identificar los riesgos y restricciones que podemos encontrar y, debido a esto, el planteamiento de estrategias alternativas.
- Evaluación y reducción de riesgos : En esta fase se evalúan las alternativas que se han identificado debido a los riesgos y se definen los pasos a seguir.
- Desarrollo y Pruebas: Tras la evaluación de las alternativas se elige una de ellas para realizar el desarrollo del sistema. Tras el desarrollo se realizan pruebas sobre el mismo para comprobar y asegurar el funcionamiento correcto.
- Planificación de la siguiente fase : En esta fase se realiza una revisión del avance del proyecto hasta el momento y se toma la decisión de continuar o no con otro ciclo de la espiral. En el caso de tomar la decisión de continuar, volveremos a comenzar por el primer punto de esta lista, mientras que si no lo hacemos, habrá finalizado el ciclo de vida del proyecto y habremos llegado al final de la espiral [27].

A continuación se muestra una imagen del desarrollo en espiral que se ha seguido en este proyecto, diferenciando entre los cuatro puntos y las fases mencionadas. Cabe destacar, que la longitud de las líneas no tienen que ver con el tiempo o la dedicación que se ha empleado para realizar las acciones de cada fase y atienden meramente a razones estéticas.

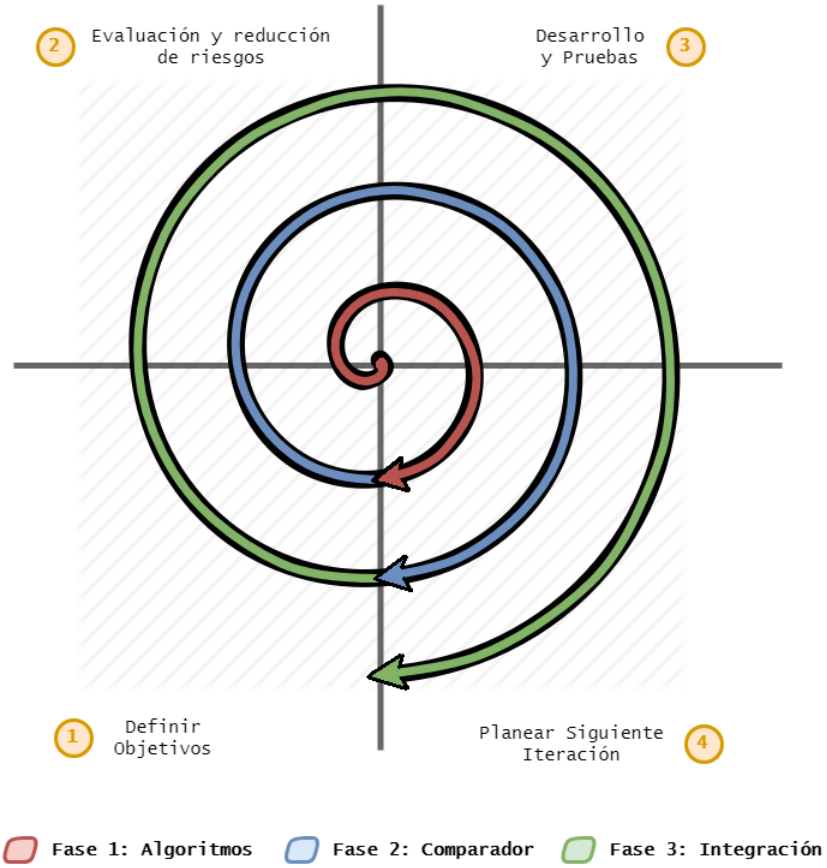

Fig. 3.9. Metodología en espiral seguida en el desempeño del proyecto

Si bien es cierto que a pesar de que estos cuatro puntos se repiten por cada iteración de la espiral, en función de la fase en la que nos encontremos se da más importancia a unas acciones u otras. Por ejemplo, al comienzo del ciclo de vida del proyecto, en la primera fase, no se dio tanta importancia a las pruebas puesto que primaba más la necesidad de investigación y análisis, mientras que en la última fase, si que se le dio más importancia debido a que la implementación realizada en dicha fase formaba parte del producto funcional final.

De esta forma las acciones catalogadas como definición de objetivos en nuestro proyecto corresponden principalmente al estudio de las tecnologías y la teoría, el estado del arte, estudio de las alternativas, establecer el alcance de cada una de las fases (que vamos a hacer y que no), etc.

En cuanto a la evaluación y reducción de riesgos, se corresponde con el estudio de las alternativas, las decisiones que se han tomado para llevar a cabo el proyecto, el diseño y la determinación de los pasos que se van a llevar a cabo para desarrollar la fase.

La tarea de desarrollo y pruebas se corresponde a la implementación del código que maneja el funcionamiento de la API desarrollada para nuestro proyecto y la posterior comprobación del correcto funcionamiento de la misma.

Finalmente, la planificación de la siguiente iteración se corresponde con la necesidad

de seguir avanzando en el proyecto. En la fase 1 y 2 se continúa puesto que aun no se han logrado los objetivos planteados mientras que en la fase 3 se determina que ya se han implementado mejoras en las herramientas que ofrece MongoDB y no se continúa con el ciclo de vida de proyecto. No obstante como se tratará en el punto *"Líneas futuras de trabajo"*, podríamos seguir iterando en la espiral y desarrollar más funcionalidades para la base de datos NoSQL MongoDB definiendo nuevas fases.

3.10. Planificación del proyecto y presupuesto

En este punto, dada la metodología que se va a realizar, se establece una planificación del proyecto a la que se intentará ser lo más fiel posible. Además también se presupuesta el coste del proyecto en función del tiempo que se estime en primer lugar, y los materiales (tanto humanos como tecnológicos) que se utilizarán a lo largo del ciclo de vida del proyecto.

3.10.1. Planificación

El objetivo que se persigue en este punto no es otro que, dada la metodología escogida y la organización en fases que se ha establecido, estimar una planificación temporal de la duración del proyecto así como dar un presupuesto inicial del trabajo.

Comenzando por la planificación temporal, en primer lugar cabe mencionar que la naturaleza de este trabajo no es otra que la de finalizar los estudios de grado de Ingeniería informática, para la cual se exige la realización de un proyecto final de grado. De acuerdo con el plan de estudios de este grado impartido en la Universidad Carlos III de Madrid, encontramos que según el plan que se está estudiando en esta universidad para dicho grado (Plan 2011) la realización del Trabajo de Fin de Grado equivale a un esfuerzo por parte del alumno de 12 créditos ECTS [28]. Siguiendo la información que ofrece la universidad [29] sabemos que un crédito ECTS equivale a 25 horas. Por tanto tenemos que el tiempo del que disponemos para realizar el desarrollo completo del trabajo es:

$$N^{o}Creditos_{TFG} \cdot Horas_{credito} = 12 \cdot 25 = 300 \quad horas$$

Puesto que el trabajo se organiza en tres fases como ya se ha mencionado, y conociendo las actividades que se deben realizar en cada fase, se ha estimado que la duración completa del desarrollo de dichas fases sean 210 horas, dejando 70 horas restantes dedicadas a la escritura de la documentación que nos ocupa y el despliegue del sistema en npm y 20 horas como colchón de tiempo para posibles problemas que puedan surgir a lo largo del proyecto. Además, se ha determinado también que el reparto de horas destinadas a cada fase se realice de forma equitativa, es decir, que todas tengan la misma duración puesto que las tareas que se deben realizar tienen la misma carga de trabajo. De esta manera cada fase tendría una duración de 70 horas.

Puesto que el comienzo del proyecto está fijado en torno a la fecha de matriculación de la asignatura, y se prevé la realización de la misma a mediados de Noviembre, estimamos que el comienzo es el día 13 de Noviembre de 2017.

La duración estimada del proyecto ha sido de 7 meses. Esto es debido a que el alumno trabaja y entre diario a penas cuenta con tiempo para la dedicación al mismo, además de que se pretende realizar la entrega en la convocatoria de Junio, teniendo esta como fecha límite el 19 de Junio. Por tanto, la fecha estimada de finalización, habiendo comenzado el

13 de Noviembre la estimamos como muy tarde, justo a 7 meses más tarde: el 13 de Junio de 2018, para no situar la fecha de finalización muy cercana a la fecha límite.

Disponiendo de estos datos, encontramos que debemos repartir las 300 horas estimadas (contando con las 20 horas que hemos destinado al colchón de tiempo puesto que en caso de que nos hagan falta, las utilizaremos) entre los 7 meses de duración establecida. Puesto que la duración de los meses es variable, se calcula la duración del proyecto en días, obteniendo que entre las dos fechas de inicio y fin se suceden 212 días.

$$\frac{Horas_{totales}}{Dias_{totales}} = \frac{300}{212} = 1,42 \quad horas/dia$$

Ahora bien, disponiendo de las horas al día que debemos trabajar para llegar a la fecha estimada, en conjunto con la duración de cada fase, se puede estimar las fechas en las que se comienza y finaliza cada una de ellas.

En primer lugar, puesto que se dispone de 20 horas destinadas a un colchón de tiempo, si se realiza el cálculo de dividir esta cifra entre las horas de trabajo diarias se obtiene como resultado la cifra de 14 días de colchón. Esto establece que, si no surge ningún inconveniente a lo largo de la realización del proyecto, la fecha de finalización del mismo debería producirse el día 30 de Mayo de 2018.

En cuanto al grueso del trabajo, se ha estimado estimado dedicar 210 horas en total al desarrollo (70 horas por fase) y 70 horas a la escritura de la documentación. Esto supone que para cada uno de estos intervalos, con las horas al día estimadas que se van a trabajar, equivale a un trabajo de aproximadamente de 49 días y medio.

Por tanto, las fechas estimadas para cada una de las fases son las que se muestran en la siguiente tabla:

Fase	Fecha Inicio	Fecha Fin
Fase 1	13-11-2017	01-01-2018
Fase 2	02-01-2018	20-02-2018
Fase 3	21-02-2018	11-04-2018
Documentación y Despliegue	12-04-2018	31-05-2018

Tabla 3.47. Fechas estimadas para cada fase del proyecto

Si se representa la duración de cada una de las fases con un diagrama de Gantt, obtenemos una visión general de la planificación a lo largo del tiempo como puede apreciarse en la figura de la página siguiente.

id	Tarea	Duración	Comienzo	Final
1	Realización completa del trabajo fin de grado	98 días	13.11.17	30.05.18
2	Fase 1: Algoritmos difusos	49.5	13.11.17	01.01.18
3	Fase 2: Comparador de documentos	49.5	02.01.18	20.02.18
4	Fase 3: Implantación en MongoDB	49.5	21.02.18	11.04.18
5	Documentación y despliegue en npm	49.5	12.04.18	30.05.18

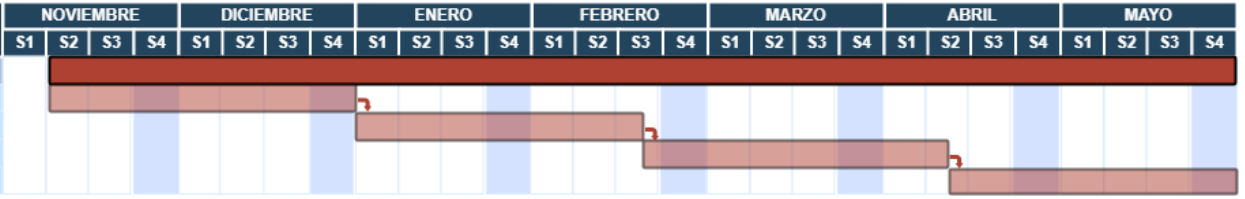

Fig. 3.10. Diagrama de Gantt de la planificación estimada

3.10.2. Presupuesto

En esta sección se procede a realizar un presupuesto del trabajo. Para ello se calculan dos tipos de costes: directos e indirectos, en función de la planificación que se ha estimado. Es decir, se calcularán los valores de los costes teniendo en cuenta la planificación descrita en el punto anterior.

Para el cálculo de los **costes indirectos** se ha tomado el valor de un 20 % sobre los costes directos.

Para el cálculo de los **costes directos** se han tenido en cuenta los siguientes aspectos:

- Costes de personal en función de las horas de trabajo que se han estimado.
- Costes de material (hardware y software).

3.10.2.1. Costes de personal

En primer lugar calculamos los costes de personal teniendo en cuenta que el equipo está compuesto por una única persona que será la que trabaje durante toda la duración del proyecto. El coste de un trabajador varía dependiendo del área a la que se dedica, la formación que posee y la zona geográfica en la que trabaje. En nuestro caso, la zona se localiza en España, Madrid, en el área de las TIC y con una titulación de "Graduado en Ingeniería Informática".

Dicho esto somos conocedores de que el puesto de una persona con estas características no es otro que un "informático junior", puesto que el equipo se compone de una persona recién titulada y con poca experiencia en el sector.

Según el Informe de infoempleo publicado por Adecco en 2017 [30], un técnico en el sector de las TIC en la comunidad de Madrid en posesión de un grado percibe un sueldo medio de 27.454 € brutos al año. Esto repartido en 14 pagas mensuales (puesto que el sueldo medio que calcula Adecco tiene en cuenta las pagas extra) equivale a 1.961 € brutos al mes.

Si bien este es el sueldo que percibe el empleado, nosotros debemos calcular el coste de contratar a dicho técnico, en otras palabras, sumar a dicha cuantía lo que la empresa ha de pagar a la seguridad social. Para ello utilizamos los baremos que pone a disposición el Ministerio de Empleo y de Seguridad Social Español [31], puesto que como se ha dicho antes, el coste del trabajador dependerá también de la zona geográfica en la que se le contrate, en este caso situada en España. Por tanto, para el cálculo final del coste del empleado siguiendo el régimen general (no se trata de un autónomo u otros casos especiales) se tienen en cuenta los siguientes valores:

- Contingencias comunes : Se establece en un 26,3 % en concepto de contribución para las pensiones.

- Desempleo : Se establece en un 5,5 % para un contrato de tipo general (es decir, un contrato que no es temporal) y un 6,7 % para otros tipos de contrato. En nuestro caso tomaremos el valor de 6,7 % puesto que el contrato tendrá una duración igual a la duración del proyecto.
- Formación Profesional : Se establece en un 0,6 % en concepto de formación para el empleado.
- FOGASA : Esta es la cobertura de despidos de empresas que están en quiebra, es decir, un fondo a nivel nacional que se encarga de pagar los despidos e indemnizaciones a los trabajadores si la empresa se declara insolvente. Se establece en un 0,2 %.
- Accidentes laborales : También se destina un porcentaje para posibles accidentes o enfermedad del trabajador. Este varía en función del puesto de trabajo del empleado. Puesto que en nuestro caso no se trata de una profesión de riesgo, se establece en un 2 %.

Por tanto, el desglose completo del coste del empleado es el siguiente:

Concepto	Valor (€/mes)
Salario Bruto mensual	1.961
Contingencias comunes	515,74
Desempleo	131,39
Formación Profesional	11,78
FOGASA	3,92
Accidentes laborales	39,22
COSTE TOTAL	**2.663,05**

Tabla 3.48. Desglose de costes de contratación de un empleado

Por tanto, el coste de personal es de 2.663,05 €/mes de los cuales extraemos el coste por hora para poder calcular el coste de personal del proyecto (ya que este se mide en horas). Puesto que el sueldo que se ha descrito arriba equivale a un sueldo de jornada completa, se tomarán de referencia las horas establecidas según el Estatuto de los Trabajadores [32]. En esta ley se recoge que no se puede superar las 40 horas de trabajo semanal, por lo que sabiendo que un mes tiene 4 semanas de media, las horas mensuales totales trabajadas son 160. Dividiendo el coste total del trabajado entre dichas horas podemos obtener el precio €/hora.

$$\frac{CosteTotal_{mes}}{Horas_{mes}} = \frac{2663{,}05}{160} = 16{,}64 \quad euros/hora$$

Por tanto, sabiendo el precio del trabajador a euro por hora trabajada y las horas de

duración que se han estimado para el proyecto, tenemos que el coste total de personal se estima en:

$$Precio_{horas} \cdot Total_{horas} = 16{,}64 \cdot 280 = 4659,2 \quad euros$$

Adicionalmente, si nos viéramos en la obligación de utilizar los 14 días (20 horas) que se han estimado como margen temporal por si surgieran inconvenientes, nos encontraríamos que tendíamos que destinar más presupuesto al proyecto. Concretamente deberíamos destinar:

$$Precio_{horas} \cdot Total_{horas} = 16{,}64 \cdot 20 = 332,8 \quad euros$$

Por tanto esta cifra se ha tenido en cuenta (al igual que en la planificación se tenía en cuenta como margen temporal) como un margen de presupuesto por si acaso finalmente la duración es más larga que lo que se ha estimado, no presupuestar el proyecto inferiormente.

3.10.2.2. Costes de Material

Una vez que hemos presupuestado los costes de personal, debemos calcular los costes de materiales. Cabe destacar que se estima que para la realización de este trabajo no se deberá emplear ningún tipo de adquisición de hardware/software o realizar algún tipo de inversión. Aun así debemos estimar el gasto de los materiales previamente adquiridos. Dividimos dichos costes en dos tablas, los ocasionados por el hardware en primer lugar, y los ocasionados por el software en segundo lugar:

Concepto	Valor (€)
iMac 21.5-inch mid-2011	1.190
Asus VivoBook E200HA	213
Ratón Logitech G402	34,99
TOTAL	**1.437,99**

Tabla 3.49. Coste completo del hardware

No obstante, no podemos imputar el gasto total del hardware al proyecto, puesto que este se encuentra adquirido con anterioridad al comienzo del mismo. Por ello se calcula el coste del hardware en función del porcentaje de la vida útil del mismo que se dedicará al proyecto.

Concepto	Coste (€)	% Uso	Días Uso	Vida útil (días) [1]	Coste imputable [2]
iMac	1.190	100	198	1825	129,12
Asus	213	100	198	1825	23,11
Ratón	34,99	100	198	1825	3,79
TOTAL	**1.437,99**				**156,02**

[1] El valor de la vida útil de un elemento de hardware se establece en 5 años.

[2] El coste imputable se calcula utilizando la amortización $(Tiempo_{usoEnProyecto}/Tiempo_{vidaUtil}) \cdot Coste_{equipo} \cdot \%Uso$

Tabla 3.50. Costes imputables del hardware

A continuación se muestran los gastos en Software, que como se puede apreciar, han sido nulos debido a que todos los programas que se han utilizado están catalogados como OpenSource y no es necesario realizar ningún tipo de inversión para su utilización.

Concepto	Valor (€)
MongoDB Standard Edition	0
Brackets	0
node.js	0
npm	0
github	0
MikTex	0
draw.io	0
TOTAL	**0**

Tabla 3.51. Coste de elementos software utilizados

Por tanto los costes de material, entre hardware y software, ascienden a **156,02 €**.

Al igual que ocurre en el caso de los costes de personal, en este caso al calcular el coste imputable de los dispositivos hardware en función de la duración del proyecto, podemos encontrarnos ante la situación en la que el proyecto dure más de lo estimado (metiéndose en el margen que se ha dejado de colchón) y que por tanto, los costes del material aumenten ligeramente puesto que se está haciendo uso del mismo durante más tiempo. Si esto sucede, calculándolo de la misma manera que en la tabla 5.20 para los 14 días de colchón establecidos, el valor de los costes materiales adicionales sería de 11,13 €.

3.10.2.3. Costes totales

Una vez que hemos calculado los costes de personal y los costes de material podemos obtener los costes directos del proyecto sumando ambos valores. Además también se suman los valores de colchón dinerario establecidos mediante el cálculo de los cos-

tes aplicado al margen temporal que se estima. De esta manera obtenemos la siguiente cuantía:

$$Costes_{Personal} + Costes_{Materiales} + Costes_{PersonalAdicional} + Costes_{MaterialAdicional}$$
$$= 4659,2 + 156,02 + 332,8 + 11,13 = 5159,15 \quad euros$$

Para calcular el coste total del proyecto, debemos calcular también los costes indirectos que habíamos estipulado con un valor del 20 % de los costes directos. Por tanto finalmente el coste se desglosa de la siguiente manera:

Concepto	**Valor (€)**
Costes directos	5159,15
Costes indirectos	1031,83
Costes totales Sin IVA	6190,98
Costes totales Con IVA	7491,08
TOTAL	**7491,08**

Tabla 3.52. Desglose del coste total

Por tanto, el presupuesto final de este proyecto aplicando un IVA del 21 % es de **SIETE MIL CUATROCIENTOS NOVENTA Y UNO COMA OCHO** euros.

4. EJECUCIÓN DEL PROYECTO

En este punto se detalla el diseño de la arquitectura del sistema a implementar. En primer lugar se presentan los pasos generales que se han seguido a la hora de realizar la implementación y posteriormente se describe modularmente para explicar con detalle todos los elementos que lo componen.

Como se ha dicho anteriormente, el sistema de expansión (como lo llamaremos a partir de ahora) está dividido en tres grandes partes, y cada una de ellas se desarrollará en una fase distinta. La primera de ellas consta de lograr la implementación de los algoritmos difusos ya mencionados en el estado del arte. La segunda consta de el desarrollo del componente que sea capaz de comparar documentos completos, apoyándose en los algoritmos implementados de la primera fase y dar un valor de similitud entre ellos independientemente de los tipos de datos que los compongan y de su estructura. Finalmente la última parte consta de implementar las funciones de inserción, recuperación y borrado haciendo uso del comparador de documentos.

Es objetivo de este punto, determinar la estructura que van a seguir las fases previamente a realizar la implementación para que dispongamos de una visión clara de como debe realizarse. Puesto que ya conocemos la tecnología en la que se va a desarrollar el sistema de expansión, en este punto solamente queda diseñar cada uno de los componentes que formarán el sistema final en base a los requisitos expuestos. No obstante, antes cabe destacar que las dos primeras fases se pueden desarrollar sin necesidad de plantearnos el diseño o la forma final de la solución, ya que ambas son pasos intermedios de obligado cumplimiento antes de comenzar la fase final del sistema de expansión, pero el desarrollo general será mucho más sencillo si disponemos de un orden y diseño iniciales.

4.1. Descripción funcional

En esta sección se presenta de forma global la arquitectura que se ha diseñado para el sistema de expansión. Como se ha especificado con anterioridad, consta de tres partes o módulos principales: El módulo de algoritmos, el módulo del comparador y el módulo de las funciones difusas.

Antes de nada, se debe mencionar que estos módulos se han integrado en forma de API. Como se verá en el siguiente punto, la forma final que han tomado los métodos implementados son muy similares a los que implementa el módulo de MongoDB que se ha utilizado para realizar la conexión. No obstante, y como se especificó en los requisitos, la funcionalidad que realiza el sistema de expansión debe ser transparente para el usuario, y ser compatible independientemente de la naturaleza de la conexión con MongoDB (Local, cluster...),la base de datos y la colección de documentos. Por tanto la API dispondrá de unas opciones configurables en las que, previamente a la utilización del sistema

de expansión se indicarán el string de conexión a MongoDB, el nombre de la base de datos y colección sobre las que se pretenden realizar las operaciones. Además, y como se mencionará en las siguientes líneas, se ha desarrollado otro método adicional al módulo comparador para establecer una similitud entre dos documentos. La opción de usar uno u otro también se verá contemplada aquí. Finalmente también se introduce como opción algunos parámetros que requiere el módulo de algoritmos para su correcto funcionamiento. La interfaz de la API, así como las opciones de las que se disponen se explicarán en el siguiente punto: *"Descripción modular"*.

El módulo de algoritmos recoge la implementación en varias clases de algoritmos de aproximación de cadenas y de pertenencia difusa, que tal y como se ha expuesto en los requisitos, cada uno de ellos será necesario para medir la similitud entre distintos tipos de dato en los documentos BSON de MongoDB. Este módulo será por tanto imprescindible para el funcionamiento correcto del sistema de expansión, puesto que la lógica que se encarga de dar la similitud que tienen dos entradas (bien sean cadenas de caracteres, números o fechas) se encuentra aquí contenida.

El módulo del comparador de documentos es sin duda una parte importante del sistema de expansión, no obstante, el funcionamiento correcto de las operaciones difusas no dependen enteramente de él, ya que estas implementan otro método de realizar la comparación a parte del uso de este módulo. Dicho método alternativo se mencionará durante la explicación del módulo de las operaciones difusas. Por lo demás, este módulo consta de una única clase la cual implementará un algoritmo que devuelve un valor de similitud entre dos documentos que se indiquen. En el siguiente punto se explicará de forma detallada cual es el funcionamiento exacto del mismo.

El módulo que implementa las operaciones difusas es el encargado de implementar las operaciones finales sobre la base de datos. Será el encargado de conectarse a MongoDB a través del driver para Node.js y utilizar la similitud entre documentos para efectuar las operaciones desarrolladas:

- En el caso de la inserción difusa, se recorrerá la colección en la que se pretende insertar, comparando el documento a insertar con los almacenados. Si la similitud calculada del documento que se inserta con alguno de los almacenados supera el umbral de similitud marcado, no se insertará el documento. Sin embargo, si ninguno de ellos supera el umbral, si se procederá a la inserción. Cabe destacar que se implementará la versión de insertar solamente un documento, o insertar varios.
- En el caso del borrado difuso, este se implementará como una manera de "limpiar" las colecciones. Se recorrerá la colección al completo fijando un documento y comparando con los demás. Si alguna de las comparaciones realizadas arroja un valor de similitud por encima del umbral, entonces se borrará dicho documento. Cuando haya finalizado la limpieza para ese primer documento fijado se fijará uno nuevo para volver a comenzar con el ciclo de limpieza y se volverá a repetir este flujo para cada documento de la colección. De esta manera, al finalizar la ejecución se habrán

eliminado los documentos que pudiesen resultar información repetida o muy similar a los ya existentes y que no aportasen valor al conjunto de datos almacenado.

- En el caso de la recuperación difusa, se recorrerá la colección buscando aquellos documentos que presenten una similitud con el documento especificado. Si supera el umbral de similitud estipulado entonces se recuperará dicho documento, mientras que si no la supera no se recuperará. Finalmente, y como se ha mencionado anteriormente, la comparación se realizará con documentos completos, por lo que si se quiere realizar una búsqueda difusa de un documento, buscando solamente por un atributo, probablemente los resultados obtenidos no sean los esperados. Esto se explicará más en profundidad en el siguiente punto, pero a grandes rasgos, lo que sucede es que si solamente se realiza la búsqueda difusa indicando un atributo, al realizar una comparación de documentos completos, el algoritmo toma el atributo indicado como un documento completo y penalizará (disminuyendo el factor de la similitud) cuando compare con documentos que seguramente posean más atributos que el indicado.

No obstante, y como se mencionó durante la explicación breve del módulo del comparador, esta clase dispondrá de una forma alternativa de calcular la similitud en la que no se usará el módulo comparador de documentos. Esta solución, simplemente tomará el documento en cuestión como una gran cadena de caracteres, (o en otras palabras: serializar el documento JSON) y comparará directamente los documentos mediante las técnicas de aproximación de cadenas. Cabe destacar que para esta opción, se permitirá al usuario que escoja entre los algoritmos de aproximación de cadenas desarrollados con cual de ellos se calculará la similitud entre los documentos serializados. En el siguiente punto, se explican de manera más profunda ambos métodos (uso del módulo del comparador o uso de la serialización de documentos) y las ventajas e inconvenientes de cada uno de ellos.

Finalmente se dispondrá de un pequeño módulo de “utilidades” en el que se implementan dos clases: una primera denominada “Shingles” que se utilizará para obtener los ngram de una cadena de caracteres y una segunda llamada “Spinner” que dibujará una barra de carga en la consola durante la ejecución de las operaciones difusas para ofrecer al usuario información sobre el estado de la ejecución puesto que, aunque se mencionan más adelante las causas, cuanta más información contenga la BBDD más se demorará la ejecución.

Para finalizar esta sección se muestra un diagrama de flujo que ilustra el funcionamiento general que seguirá el sistema.

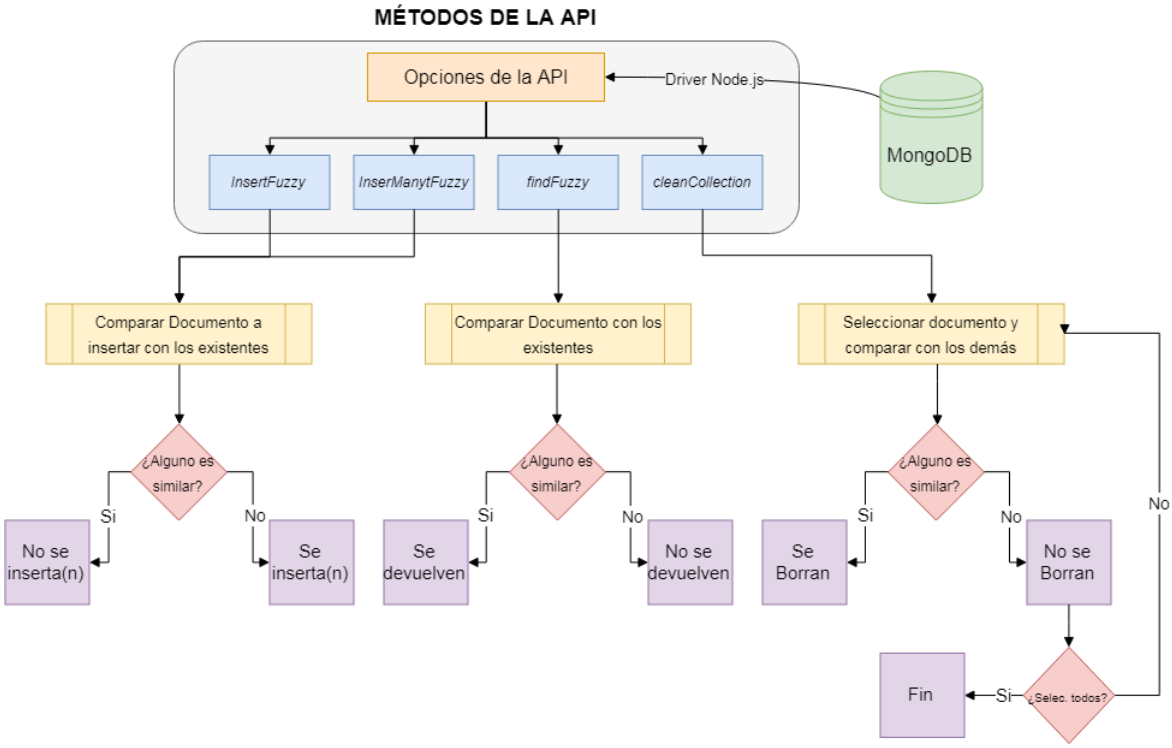

Fig. 4.1. Diagrama de flujo general de la aplicación

4.2. Descripción modular

En esta sección se presenta de forma detallada cada uno de los módulos introducidos anteriormente. El orden establecido para la explicación ha sido el mismo que se siguió para el desarrollo del sistema; por tanto primero se explicará el módulo de los algoritmos, seguidamente del módulo del comparador de documentos y finalmente del módulo de las operaciones difusas. Adicionalmente se menciona al módulo de utilidades cuando realiza intervenciones en el módulo de algoritmos y en el de las operaciones difusas, pero no se le dedicará una sección como a los otros módulos.

4.2.1. Módulo de algoritmos

Como se ha ido explicando en numerosas secciones antes de llegar aquí, este se trata del módulo que implementa los algoritmos de aproximación de cadenas y pertenencia difusa.

Como se especifica en los requisitos expuestos en el capítulo de *"Definición del proyecto"* el sistema de expansión deberá disponer de estos algoritmos implementados para establecer la similitud entre cadenas de caracteres, números y fechas (Algunos de los tipos de dato que ofrece MongoDB). Para obtener la similitud del resto de tipos de dato, se explicará en el punto siguiente, que es el que atañe al comparador de documentos, pero la base sobre la que se asienta la obtención de la similitud del resto de tipos de datos no es otra que la de los algoritmos que se implementan en este módulo aunque con leves modificaciones.

Los algoritmos que se han implementado por tanto han sido los que se exponen a continuación, cada uno en una clase distinta. Debemos destacar que todos los algoritmos referentes a la aproximación de cadenas se han implementado tomando como referencia una librería de Java [5] y modificando el código para que se adapte al lenguaje de programación JavaScript.

Antes de comenzar la explicación de cada una de las clases cabe destacar que, como se expuso en el estado del arte en la sección *"Aproximación de cadenas en NoSQL"*, existen algoritmos de aproximación de cadenas que utilizan la teoría de los ngram para calcular la similitud. Puesto que esto sucede en tres de los algoritmos que se han implementado y que veremos más adelante (Jaccard y Kondrak), se ha extraído la funcionalidad de, dada una cadena partirla en los ngram, en el módulo "Utils" a través de la clase "Shingles".

[5] `https://github.com/tdebatty/java-string-similarity`

4.2.1.1. Levenshtein

Esa clase implementa el algoritmo de Levenshtein, que se utiliza para comparar la similitud entre dos cadenas de caracteres. Para ello se crea un constructor llamado *Levenshtein* que recibe por parámetro dos cadenas de caracteres (aquellas que se van a comparar). Esta clase implementa tres métodos para conseguir el objetivo de calcular la similitud:

- *getDistance()* : El propio del algoritmo de Levenshtein. Este algoritmo no implementa una medida de similitud, si no del factor opuesto, la distancia. Por tanto se ha creado un método que dadas dos cadenas de caracteres calculan la distancia entre ellas. Así, da un valor numérico dadas las inserciones, borrados y sustituciones que hay que realizar sobre la cadena *A* para transformarla en la cadena *B*, por lo que cuanto mayor sea este valor, más distintas serán las cadenas entre sí.

- *getDistanceNormalized()* : Este método simplemente realiza una llamada al método de *getDistance()* y normaliza el valor entero de la distancia entre el intervalo [0, 1]. Así dos cadenas que sean iguales adquirirán el valor de 0 mientras que otras que no se parezcan en nada tendrán el valor de 1, pareciéndose más al modelo de lógica difusa que también utiliza el rango entre 0 y 1 para establecer la pertenencia a un conjunto. En este caso el conjunto no es otro que la distancia entre las dos cadenas.

- *getSimilarity()* : Este método es el que se utilizará finalmente, ya que realizará una llamada al método anterior *getDistanceNormalized()* y devolverá la operación inversa. Así, en lugar de devolver la distancia con valores comprendidos en el intervalos [0, 1], se calcula la similitud como la unidad menos la distancia y de esta manera cuando se obtengan valores cercanos a 0 nos indicará que las dos cadenas no se parecen mientras que los valores cercanos a 1 indicarán que ambas cadenas son prácticamente iguales.

Por tanto, cuando se utiliza el algoritmo de Levenshtein, se crea un objeto a través del constructor, indicando las dos cadenas que se pretenden comparar y se utilizará el método *getSimilarity()* para establecer la similitud entre ambas.

4.2.1.2. Jaccard

Este algoritmo utiliza la técnica ya mencionada de dividir las cadenas en secciones más pequeñas que se implementa a través de la clase “Shingles”. Por esto, además de las cadenas que pretenden compararse recibe por parámetro el valor numérico del número de caracteres que contiene cada ngram. Por tanto, sobre la cadena *Ainara* si se indica un valor de 3 para el ngram, esta quedará dividida en los ngrams: “Ain”, “ina” “nar”, “ara”; mientras que si el valor indicado es 2 la división es: “Ai”, “in”, “na”, “ar”. “ra”.

Esta clase implementa el método *getSimilarity()* el cual primero divide las cadenas en los ngrams invocando a la clase “Shingles” del módulo “Utils” y luego aplica la siguiente

fórmula

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

que fue anteriormente expuesta en el estado del arte, siendo A y B el conjunto de ngrams de cada una de las cadenas respectivamente.

Al igual que sucedía con el algoritmo de Levenshtein, este algoritmo implementa el método *getSimilarity()* aplicando esta fórmula sobre los ngrams. Así de esta manera si la cadena A es el nombre "Ainara" y la cadena B el nombre "Ainhoa", con un valor de 3 para el ngram se generan los siguientes conjuntos:

$$A = [\prime Ain\prime, \prime ina\prime, \prime nar\prime, \prime ara\prime] \qquad B = [\prime Ain\prime, \prime inh\prime, \prime nho\prime, \prime hoa\prime]$$

Por tanto el valor de similitud entre ambas cadenas es el número de ngram comunes que poseen ambas cadenas dividido entre el número de ngrams distintos totales. En este caso comparten un ngram por lo que la similitud será:

$$1/7 = 0,14$$

Como puede observarse, las que a simple vista son dos cadenas que tienen parecido entre sí cuando aplicamos este algoritmo no se obtiene muy buenos resultados. Esto es principalmente debido a que los algoritmos que hacen uso de la partición de cadenas en ngrams suelen ser más óptimos para comparar cadenas largas. Aunque como se verá en el módulo del comparador de documentos, este algoritmo en cuestión, se ha implementado con el objetivo de aplicarlo a la obtención de la similitud entre arrays.

4.2.1.3. Kondrak

Este algoritmo concreto no se ha mencionado en el estado del arte, pero si se ha implementado debido a la gran utilidad que se le ha visto tras la lectura del paper escrito por Kondrak(que da nombre al algoritmo) y en el que se expone el funcionamiento del mismo [33]. Este algoritmo en cuestión, utiliza también la partición de cadenas en ngrams, ya que en rasgos generales, lo que Grzegorz Kondrak postula en el estudio que realiza es mezclar los algoritmos de edición de distancias con los basados en ngrams y de esta manera obtener mejores resultados a la hora de realizar comparaciones entre cadenas. El algoritmo en cuestión también se propone en forma de pseudocódigo en el estudio de Kondrak, y como se ha mencionado anteriormente, se implementa en Java en la librería que se ha tomado como referencia.

Por tanto, para esta clase se tendrá de nuevo un constructor que recibe por parámetro el valor del tamaño de los ngrams y las cadenas que pretenden compararse. En el método denominado *getSimilarity()* se implementa el código propuesto por Kondrak para obtener

la similitud entre dos cadenas que devolverá un valor cercano a 0 cuando las cadenas no tengan mucho parecido y un valor cercano a 1 cuando si lo tengan.

Aunque se dedicará un capítulo completo a la evaluación del sistema, cabe destacar que cuando se realizaron pruebas con este algoritmo, fue sin duda alguna el que mejor resultado originó. A diferencia de lo que ocurre con el índice de Jaccard explicado en el punto anterior, el uso de ngrams en este caso no penaliza el valor de similitud para cadenas cortas, originando un valor similar al producido por el algoritmo de Levenshtein. Además, en cadenas de gran longitud se obtuvieron también buenos resultados.

4.2.1.4. Gauss

Al contrario que las tres clases expuestas anteriormente, las cuales aplican algoritmos de aproximación de cadenas para la comparación de cadenas de caracteres, la clase denominada *Gauss* recae en el ámbito propio de la lógica difusa, concretamente en las funciones de pertenencia difusa.

Se ha implementado esta clase con el objetivo de obtener la similitud entre dos números o dos fechas, los cuales también pertenece a los tipos de dato que ofrece MongoDB. Para ello se ha utilizado la función Gaussiana, que establece la pertenencia o no de un valor al conjunto a través de la campana de Gauss como se observa en la imagen.

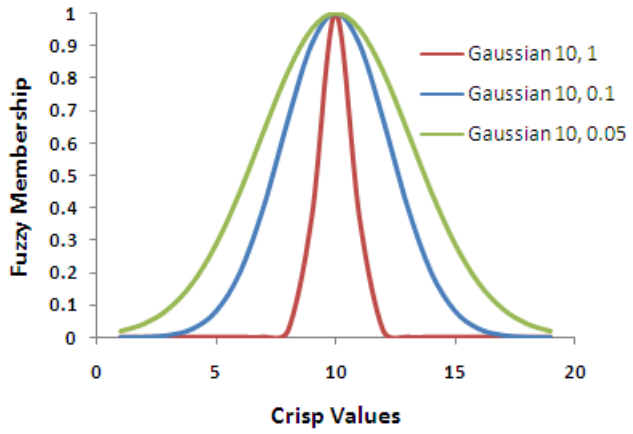

Fig. 4.2. Función Gaussiana de pertenencia difusa [34]

Y cuya función viene determinada por la aplicación de la siguiente fórmula:

$$\mu_A(x) = e^{-k(x-m)^2}$$

Dónde el valor de k no es otro que el de la apertura de la campana de Gauss, de forma que cuanto mayor sea este valor, mas estrecha será la campana, y el valor m el valor medio de la campana. En el caso de la figura mostrada más arriba, tenemos tres ejemplos en los que el valor medio se sitúa en 10 y el valor de k en los valores de 1, 0,1 y 0,05, pudiéndose observar como dicha campana se estrecha cuando el valor disminuye.

Como se puede observar el valor medio adquiere la pertenencia de 1 en el eje cartesiano *y* indicando una pertenencia total al conjunto, y según los valores se alejan de 10 tanto por arriba como por abajo esta pertenencia disminuye. Al ser *k* el valor que marca la anchura de la campana, dispondremos de más o menos margen de pertenencia en función del valor que se le otorgue.

Esto aplicado al caso de realizar comparaciones de similitud entre números, establecemos como valor medio aquel con el que queremos comparar y un valor para *k* que variará en función del dominio de los datos, ya que no es lo mismo realizar una comparación entre peso medido en Kilogramos o en gramos. Para el dominio de Kg, si estamos hablando del peso de una persona, necesitaremos una campana estrecha puesto que si no, enseguida dejan de ser similares, mientras que si medimos en gramos, la campana deberá ser más ancha puesto que si no solamente tomará como similares pesos que difieran muy poco o nada del original. Teniendo esto en cuenta, se decidió que a la hora de utilizar este algoritmo para comparar los documentos, fuese el propio usuario de la API el que introdujese el valor de *k*, ya que será el conocedor del dominio sobre el que se aplicará el sistema de expansión.

En cuanto al caso de utilizar la función de pertenencia para las fechas, en realidad lo que se ha realizado ha sido un paso del valor de la fecha a milisegundos gracias a una función que ofrece JavaScript y convertirlo en días. De esta manera, y como ocurría en el caso anterior el valor de *k* marcará si tomamos como fechas similares que se trate del mismo día o damos un poco más de margen ensanchando la campana y tomamos como fecha similar otra que se haya sucedido 10 días antes.

Al igual que ocurría con los métodos anteriores, esta funcionalidad se ha incluido en un método llamado *getSimilarity()* para la clase *Gauss*.

4.2.1.5. Algorithm

Esta clase es la encargada de unir todos los algoritmos anteriormente expuestos a excepción del algoritmo que implementa la pertenencia difusa con la campana de Gauss y se ha codificado como alternativa al uso del comparador de documentos.

Como ya se ha mencionado anteriormente, la comparación de los documentos puede realizarse, además de con el módulo del comparador, mediante la serialización de los documentos y aplicando uno de los algoritmos de aproximación de cadenas a la cadena que se genera al convertir un JSON a String. Es para este caso para el que se ha concebido esta clase, ya que el usuario podrá decidir que algoritmo utilizar para realizar obtener la similitud entre los documentos serializados. Por tanto esta clase posee un constructor que recibe por parámetro el nombre del algoritmo: 'levenshtein', 'jaccard' o 'kondrak' y el valor de los ngram en el caso de necesitarlo. Implementa por tanto un método *getSimilarity()* que en función de que algoritmo se haya indicado ejecutará la obtención de similitud de uno u otro. La desventaja de utilizar este método para obtener la similitud es que tiene mucha

menos precisión que el del módulo de documentos debido a que al comparar el documento completo convertido en cadena tiene en cuenta todo, incluido los caracteres de separación como las llaves, comas... lo cual puede añadir ruido a la similitud. Sin embargo esto no sucede así con el comparador de documentos, que es capaz de comparar cada tipo de dato de acuerdo con las características que posee. A cambio, el proceso puede resultar mucho más lento con el comparador de documentos como veremos en el siguiente punto.

4.2.2. Módulo de comparación de documentos

Este módulo, como se ha explicado en anteriores ocasiones, es el encargado de realizar la comparación de documentos. Cabe destacar que dicha comparación la podemos realizar de dos formas distintas. Una de ellas es mediante el uso de este módulo, y la otra como ya se ha mencionado, mediante la serialización del documento. En esta sección nos limitaremos a explicar la que atañe a dicho módulo ya que el funcionamiento de la comparación mediante la serialización se ha explicado anteriormente.

No obstante adelantamos la razón de que existan dos maneras distintas de comparar documentos y esta no es otra que el tiempo de ejecución. Como se verá más adelante en este punto, el algoritmo que se ha implementado para la comparación de documentos es recursivo, puesto que si no seríamos incapaces de gestionar la casuística de los documentos anidados que permite MongoDB. Esto, unido a que para realizar las operaciones difusas se debe recorrer la colección completa sobre la que se está operando como se ha mencionado previamente, hace que si el tamaño de dicha colección es muy grande, y los documentos almacenados en ella poseen muchos documentos anidados, el tiempo que tarda el sistema de expansión en realizar la operación difusa en cuestión se dispare. Asi que debido a esto, nos vimos en la obligación de proveer de otra alternativa para la realización de las comparaciones entre documentos.

Dicho esto procedemos a la explicación del funcionamiento de este módulo no sin antes mencionar que nos ha supuesto un gran desafío su desarrollo ya que había que manejar muchas posibilidades al tener los datos una forma tan desestructurada.

Como ya hemos mencionado, lo primero que hay que conocer de este módulo es que se compone de una única clase llamada *DocumentCompare* que implementa varios métodos mediante los cuales se consigue el objetivo propuesto. El constructor de esta clase recibe por parámetro los valores de n (el número que indica el tamaño de ngram utilizado en los algoritmos de aproximación de cadenas) y k (el número que indica la anchura de la campana de Gauss), además de un valor de umbral de similitud. Este último se utiliza para determinar la similitud de los arrays como explicaremos más adelante.

Una vez expuestos los parámetros de entrada que utiliza este módulo, comenzamos con la explicación de su funcionamiento. Lo primero de todo es conocer cómo sabe el algoritmo, disponiendo de dos documentos A y B que atributos comparar de cada uno de ellos. La solución por la que se ha optado ha sido bastante sencilla: puesto que los

documentos no son más que una composición de datos con la organización clave-valor, se comparan aquellos que en el documento *A* poseen la misma clave que en el documento *B* y además el valor de ambos coincide en el tipo de dato. De esta manera si poseemos los documentos:

```
Documento A = {
    "nombre" : "Ainara",
    "edad" : 22,
    "estudios" : ["ESO","Bachillerato","Grado"],
    "localidad" : "Carranque, Toledo"
}

Documento B = {
    "nombre" : "Miguel",
    "edad" : 23,
    "estudios" : "ESO,Bachillerato,Grado",
    "vivienda" : "Alcorcon, Madrid"
}
```

Se realizará la comparación entre el atributo “nombre” y “edad” al tener el mismo nombre para la clave y el mismo tipo de dato (String y numérico respectivamente), pero no lo hará para el atributo “estudios” a pesar de que en ambos documentos existe esa clave puesto que no se trata del mismo tipo de datos (en uno es un array y en otro un String) y tampoco lo hará para el campo “localidad” porque en el documento *B* ni si quiera existe esta propiedad.

Por tanto es de esta manera como se realizan las comparaciones para un documento. Si bien de esta manera obtenemos la similitud de los atributos de forma independiente, por lo que para dar un valor final de similitud para el documento completo simplemente se realizará una suma de todas las similitudes y se dividirá entre el número total de atributos. Además, se penalizará al valor de similitud para el caso de atributos que no existan en un documento o en otro. De manera que suponiendo que para el ejemplo expuesto la similitud entre los documentos para el atributo “nombre” fuera de 0,4 y para la edad 0,8 el valor total de similitud no será el que viene dado por la fórmula $(0{,}4 + 0{,}8)/2 = 0{,}6$, si no que también se añadirá al divisor aquellos documentos que no pudieron compararse por inexistencia o por no coincidencia del tipo de dato, es decir, cuentan como si tuvieran una similitud igual a 0. De esta manera la operación realizada es $(0{,}4 + 0{,}8)/5 = 0{,}24$ puesto que el atributo “localidad” no existe en el documento *B*, el atributo “vivienda” no existe en el documento *A* y el atributo “estudios” no coincide en tipo. De esta forma penalizamos un valor que habría sido de una similitud del 0,6 disminuyéndolo a 0,24. Esta funcionalidad queda recogida en un método llamado *getSimilarity()*.

Una vez explicado como funciona el comparador a nivel de documento, procedemos a explicar cuáles han sido las decisiones que se han tomado para hallar la similitud para cada tipo de dato, haciendo especial hincapié en los que resultaron más complejos: Los arrays y los documentos anidados.

- Cadenas de caracteres: Aquellos atributos del documento que sean cadenas de caracteres se realiza su comparación aplicando los algoritmos de aproximación de cadenas. Concretamente para los atributos que contengan solamente una palabra se utiliza el algoritmo de Levenshtein ya que este es muy bueno con palabras cortas, mientras que con atributos que contengan más de una palabra se utilizará el algoritmo de Kondrak.

- Números y Fechas: Aquellos atributos del documento que sean Números o fechas se utilizará el algoritmo de Gauss para determinar su similitud.

- Documentos anidados: Cuando los atributos son documentos anidados, simplemente realizamos una llamada recursiva al método que calcula la similitud del documento *getSimilarity()* que calculará la similitud entre ambos documentos como se ha explicado anteriormente y este valor será el utilizado para calcular la similitud total del documento. No obstante esta implementación resulta muy penalizadora en tiempo, ya que la recursividad es un factor conocido por empeorar la velocidad de la ejecución, pero no podíamos tomar otro diseño alternativo puesto que MongoDB permite hasta 100 niveles de anidamiento en los documentos, y de no haberlo resuelto de esta forma, habríamos tenido que codificar de forma manual 100 niveles de condiciones en las que se extrajese la similitud de los documentos anidados.

- Arrays: Para calcular la similitud de los arrays y como se ha mencionado en el punto anterior, se ha usado el algoritmo de Jaccard pero aplicándolo de forma difusa.

  El algoritmo original para aplicar la fórmula que lo caracteriza, utiliza la pertenencia o no de un elemento a un conjunto. Por ejemplo, tenemos los elementos $[A, B, C, D]$ y $[A, C, D, E]$. El algoritmo establece que la similitud de ambos viene determinada por la cardinalidad de la intersección de los conjuntos (es decir, el número de elementos que comparten ambos conjuntos), en este caso 3, dividido entre la unión de ambos, que resulta ser 5. Por tanto la similitud será $3/5 = 0{,}6$.

  Esta misma operación ha sido la que se ha realizado para los arrays, modificándola ligeramente para calcular la intersección de forma difusa. Aquí surge una de las primeras dificultades, y es que los arrays en MongoDB pueden contener cualquier tipo de dato, incluídos documentos. Así que para la realización de este algoritmo se ha creado un método al que se ha denominado *ContainsFuzzy(array, element)*.

  Este método recibe el array (al igual que los presentados en el ejemplo solo que contendrán los datos típicos de MongoDB) y el elemento del que se quiere comprobar si la pertenencia al array. Por tanto se recorre dicho array comparando el tipo del elemento con el que se está recorriendo en el array y si es coincidente, se calcula la similitud con el mismo. Si la similitud supera el umbral (parámetro que ya habíamos mencionado que se utilizaba para el cálculo de la similitud de los arrays) entonces dicho elemento formará parte del conjunto de la intersección. Si aplicamos este método para la comparación entre arrays, sabremos el tamaño del

conjunto intersección de ambos y podremos aplicar la fórmula de jaccard para obtener la similitud de los arrays, funcionalidad que se recoge en un método llamado *getArraySimilarity()*.

Cabe destacar que esto se ha realizado para que funcione con cualquier tipo de dato que contenga el array; esto incluye los documentos, en cuyo caso se volverá a realizar una llamada recursiva a *getSimilarity()*, y los arrays, caso en el que se llamará recursivamente a *getArraySimilarity()*. Por tanto y al igual que sucedía con los documentos anidados, esto supone un coste en tiempo si se nos presentan estos casos, ya que el algoritmo aumentará considerablemente su tiempo de cómputo.

Finalmente destacamos sobre la implementación de este algoritmo que no se implemento esta versión "especial" de jaccard en la clase pertinente del mismo nombre puesto que a la hora de la implementación presentaba problemas de redundancias cíclicas. El objeto Jaccard del módulo de algoritmos era utilizado en el módulo del comparador y a su vez, el objeto del comparador en el Jaccard para cubrir la casuística de que en el array se encontrasen almacenados Documentos.

Con esto concluiría la implementación del comparador de documentos, destacando sobre todo que se trata de una implementación que resultó todo un desafío por la gran cantidad de posibilidades que se presentaban a la hora de realizar la comparación entre dos documentos, y que finalmente se abordó como se ha descrito anteriormente.

Finalizamos la descripción destacando el hecho de que la penalización sobre los atributos inexistentes como se ha mencionado antes hace que la comparación deba realizarse teniendo en mente que se va a tener en cuenta el documento completo. Esto afecta sobre todo a la operación de recuperación difusa, ya que cabría esperar que, al igual que sucede en Mongo, si realizamos una búsqueda por una parte del documento indicando solamente algunos de sus campos, la recuperación difusa la realice únicamente teniendo en cuenta los campos indicados. Pero debido a la penalización que supone que no se indiquen los campos a la hora de realizar la comparación, seguramente no obtendríamos el resultado esperado. Por ejemplo si disponemos de una colección que almacena personas con al estructura del Documento A y pretendemos obtener los documentos similares, indicando solamente el nombre, el comparador solamente va a poder realizar la comparación con el atributo "nombre" penalizando que no aparezcan el resto, por lo que los resultados en similitud serán bajos.

4.2.3. Módulo de operaciones difusas

Este es el último módulo y el que implementa las operaciones difusas que se han ido mencionando a lo largo de este documento. Para la implementación se ha utilizado el framework de Node.js por lo que la inserción, el borrado y la recuperación difusa se implementan como funciones asíncronas, al ser esta una característica del framework. Además

se utiliza el driver de Mongo para Node.js para realizar la conexión y las operaciones pertinentes con la base de datos.

Sumado a todo esto, el uso final del sistema de expansión debe obedecer a una API implementada para el ecosistema npm, por lo que tenemos que este módulo se compone de dos clases: la clase MDBFuzzy en la que se implementa la lógica de las operaciones difusas, y la clase Index, en la que se realiza la estructuración de los métodos de la API en conjunto con las opciones configurables (los parámetros *n* y *k*, la conexión con MongoDB... entre otras) y será la que responda a las peticiones del usuario cuando la utilice.

4.2.3.1. MDBFuzzy

Este módulo es el final que implementa aquellos objetivos que persigue este proyecto. Los métodos implementados finalmente son 4: dos atienden a la inserción difusa, uno para la limpieza de las colecciones y otro para la recuperación de documentos difusa.

El constructor de esta clase recibe por parámetro el string de conexión con la base de datos de MongoDB, mientras que el resto de métodos reciben, el nombre de la base de datos, el nombre de la colección, el umbral para el cual un documento se considera similar (indicado con valores de 0 a 1), el valor de *n* y *k* para el funcionamiento del módulo de Algoritmos de forma correcta, si se quiere utilizar la opción del módulo de algoritmos y de parámetro opcional el nombre del algoritmo que se quiera utilizar para la comparación serializada con uno de los valores posibles (‘jaccard’, ‘levenshtein’ o ‘kondrak’).

En el caso de la inserción difusa, se implementan los métodos *insertOneFuzzy()* e *insertManyFuzzy()*. El primero de los dos, además de los parámetros indicados, recibe el documento que se quiere insertar y devuelve el valor 1 si se ha insertado el documento y 0 si no lo ha hecho. En caso de error devolverá dicho error. En el segundo, en lugar de recibir un solo documento, recibe un array de documentos y devuelve el número de documentos que se han insertado (0 si no se ha insertado ninguno). Ambos métodos realizan la conexión a la base de datos que se haya configurado y recorren la colección localizada en la base de datos indicada documento a documento. Utiliza el comparador o la serialización para establecer un valor de similitud entre el documento (o documentos) que se pretende insertar y el que se está recorriendo en ese momento. Si supera el umbral indicado entonces se inserta, mientras que si no lo supera se pasa a analizar el siguiente documento de la colección hasta finalizar.

En el caso del borrado, se implementa el método *cleanCollection()*, que solamente recibe por parámetro los elementos que se han indicado en el párrafo superior y devuelve el número de elementos que se han borrado (0 si no se ha borrado ninguno). Puesto que se trata de realizar una limpieza de los documentos almacenados en la colección, el algoritmo comienza fijando el primer documento que encuentra y compara este con el resto mediante el módulo comparador de documentos o la serialización. Si el valor de similitud calculado supera el umbral, entonces dicho documento se borra y se continúa inspeccionando el

resto hasta llegar al final de la colección. Una vez alcanzado el final se fija otro documento y se repite el proceso una y otra vez hasta que haya fijado todos los documentos de la colección y haya comparado si existen documentos similares a ellos.

En el caso de la recuperación de documentos, se implementa el método *findSimilarDocuments()* el cual recibe por parámetro el documento que se busca. Recorrerá la colección comparando los documentos existentes con el indicado y todos aquellos que superen el umbral de similitud los almacenará en un array. Posteriormente se devolverá este array que estará vacío si no se encontró ningún documento y si no contendrá aquellos que son similares al indicado.

Cabe destacar que todos estos métodos además implementan una característica de JavaScript muy extendida, el uso de un callback. Esto no es más que la posibilidad de indicar por parámetro una función en la llamada de los métodos anteriormente descritos. La razón de realizar así el diseño es debido a la naturaleza asíncrona de Node.js. No somos capaces de devolver los valores descritos para cada función sin implementar el callback, por lo que finalmente se decidió utilizar, además para comodidad del usuario que use la API como se verá en el punto siguiente.

Para finalizar, mencionar que dado el tiempo que le puede llevar al algoritmo finalizar la ejecución, se implementó una clase "Spinner" localizada en el módulo "Utils" que dibuja en la consola una barra de carga. De esta manera se indica al usuario que la ejecución está siguiendo su curso normalmente y que no se sucede ningún error.

4.2.3.2. Index

Esta clase es muy simple y es la que actúa como ventana final al usuario que utilizará la API.

Puesto que en la clase anterior, para la utilización de los métodos se utilizaba gran cantidad de parámetros, se ha decidido simplificar esto añadiendo a la clase index un constructor que no es más que un JSON al que se le indicarán las opciones siguientes:

- connection_string: El string de conexión a MongoDB siguiendo la documentación del driver de Mongo para Node.js [6]
- database: Nombre de la base de datos.
- collection: Nombre de la colección.
- threshold: Umbral para el cual se considera un documento similar.
- accurate: Indica si se utiliza el módulo comparador de documentos (true) o no (false)

[6] `https://docs.mongodb.com/manual/reference/connection-string/`

- algorithm : Indica el nombre del algoritmo que se utilza en el caso de utilizar la comparación por serialización (‘jaccard’, ‘levenshtein’ o ‘kondrak’).
- n : Valor numérico que indica el número de caracteres para construir los ngram.
- k : valor numérico que indica la anchura de la campana de Gauss para la obtención de la similitud entre fechas y números.

Por tanto un ejemplo del JSON de opciones sería el siguiente:

```
{
    "connection_string" : ""mongodb://localhost:27017",
    "database" : "test",
    "collection" : "collection1",
    "threshold" : 0.87,
    "accurate" : false,
    "algorithm" "kondrak",
    "n" : 3,
    "k" : 2
}
```

Fig. 4.3. Ejemplo de JSON de opciones para la API

Por tanto, en la clase index se implementarán los métodos *insertOneFuzzy()*, *insertManyFuzzy()*, *cleanCollection()* y *findSimilarDocuments()* con una estructura exactamente igual a la de los métodos que ofrece el driver de Node.js; recibiendo por parámetro el documento o documentos (si lo necesitase ya que en el caso del borrado no es necesario) y el callback mediante el cual se puede obtener el resultado una vez la ejecución haya acabado (o los errores de haberlos) y realizar con ese resultado las operaciones que sean pertinentes. El resto de parámetros que necesita la clase “MDBFuzzy” para funcionar se tomarán del JSON de opciones, simplificando así en gran medida la utilización de los métodos.

5. DESCRIPCIÓN DE RESULTADOS

En este capítulo se recogen los resultados tanto de validación, como de ejecución que se han obtenido tras la realización de este proyecto. En los resultados de validación se exponen las pruebas que se han realizado sobre el sistema de expansión presentado en el capítulo anterior y que ya fueron establecidas en el plan de pruebas del capítulo del *"Definición del proyecto"*. En el caso de los resultados referentes a la ejecución del proyecto, se expone la ejecución temporal y el coste del mismo y se compara con la planificación y el presupuesto realizados previamente.

5.1. Validación

Para verificar que el sistema desarrollado funciona perfectamente, se estableció un plan de pruebas en el capítulo del análisis en el que determinamos que las pruebas que debía superar nuestro sistema eran: Pruebas unitarias, de integración y de sistema.

En los siguientes apartados, organizados según los tipos de prueba, se describe en forma de tablas los resultados obtenidos para las pruebas que se han realizado sobre el sistema. La tabla que se ha usado para representar cada prueba ha sido la siguiente:

P/(U-I-S)-XX	
Módulo :	*Módulo(s) afectado(s)*
Métodos :	*Método(s) afectado(s)*
Parámetros :	*Parámetros de entrada en las opciones de la API*
Resultado obtenido :	*Resultado obtenido*

Tabla 5.1. P/(U-I-S)-XX: Plantilla de pruebas

Dónde los campos de las tablas indican lo siguiente:

- **P/(U-I-S)-XX** : Identifica si se trata de una prueba unitaria cuando adquiere el valor PU, una prueba de integración con el valor PI o una prueba del sistema con el valor PS. XX indica la numeración de cada tipo de prueba. Ambos valores en conjunto forman el identificador de la prueba. Estos valores coinciden para las pruebas establecidas en el capítulo de *"Definición del proyecto"*.
- **Módulo** : Módulos que se ven afectados por esta prueba.
- **Métodos** : Métodos que se ven afectados por esta prueba.

- **Parámetros** : Valores que se introducen para el umbral, el valor de n y k para que se pueda realizar la prueba correctamente.
- **Resultado obtenido** : Resultado que ha arrojado la prueba tras su realización.

5.1.1. Pruebas Unitarias

PU-01	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Levenshtein
Parámetros :	-
Resultado obtenido :	Se obtiene un resultado de similitud de 0.83, es decir, ambas palabras son similares.

Tabla 5.2. Ejecución PU-01: Algoritmo de Levenshtein: resultado positivo

PU-02	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Levenshtein
Parámetros :	-
Resultado obtenido :	Se obtiene un resultado de similitud de 0.14, es decir, ambas palabras no son similares.

Tabla 5.3. Ejecución PU-02: Algoritmo de Levenshtein: resultado negativo

PU-03	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Jaccard
Parámetros :	$n = 2$
Resultado obtenido :	Se obtiene un resultado de similitud de 0.76, es decir, ambas cadenas son similares.

Tabla 5.4. Ejecución PU-03: Algoritmo de Jaccard: resultado positivo

PU-04	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Jaccard
Parámetros :	$n = 2$
Resultado obtenido :	Se obtiene un resultado de similitud de 0.086, es decir, las cadenas no son similares.

Tabla 5.5. Ejecución PU-04: Algoritmo de Jaccard: resultado negativo

PU-05	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase kondrak
Parámetros :	$n = 2$
Resultado obtenido :	Se obtiene un resultado de similitud de 0.88, es decir, las cadenas son similares.

Tabla 5.6. Ejecución PU-05: Algoritmo de Kondrak: resultado positivo

PU-06	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase kondrak
Parámetros :	$n = 2$.
Resultado obtenido :	Se obtiene un resultado de similitud de 0.35, es decir, las cadenas no son similares.

Tabla 5.7. Ejecución PU-06: Algoritmo de kondrak: resultado negativo

PU-07	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Gauss
Parámetros :	$k = 4$
Resultado obtenido :	Se obtiene un resultado de similitud de 0,82, es decir, los números son similares.

Tabla 5.8. Ejecución PU-07: Algoritmo de Gauss: resultado positivo

PU-08	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Gauss
Parámetros :	$k = 1$.
Resultado obtenido :	Se obtiene un resultado de similitud de 0.044, es decir, los números no son similares.

Tabla 5.9. Ejecución PU-08: Algoritmo de Gauss: resultado negativo

PU-09	
Módulo :	Utils
Métodos :	*getShingles()* de la clase Shingles
Parámetros :	$n = 3$
Resultado obtenido :	El resultado obtenido ha sido exactamente igual al esperado.

Tabla 5.10. Ejecución PU-09: Fragmentar cadena de ngrams

PU-10	
Módulo :	Utils
Métodos :	*Spinner()* de la clase Spinner
Parámetros:	-
Resultado obtenido :	El resultado obtenido ha sido exactamente igual al esperado.

Tabla 5.11. Ejecución PU-10: Mostrar Spinner en la pantalla

PU-11	
Módulo :	Comparador
Métodos :	*getArraySimilarity()* de la clase Comparador
Resultado esperado :	$k = 4 \quad n = 3 \quad threshold = 0{,}85$
Resultado obtenido :	Se obtiene un resultado de similitud de 0.5, tal y como se esperaba

Tabla 5.12. Ejecución PU-11: Algoritmo de similitud para arrays

5.1.2. Pruebas de integración

PI-01	
Módulo :	Comparador de documentos
Métodos :	*getSimilarity()* de la clase DocumentCompare
Parámetros :	$k = 2 \quad n = 3 \quad threshold \quad 0{,}6$
Resultado obtenido :	Se obtiene un resultado de similitud de 0,72, es decir, los documentos poseen cierta similitud. No se obtiene un resultado mayor porque para el documento de "Dennisse Ritchson" hay dos atributos que no se han especificado que si posee el otro documento, y como ya explicamos con anterioridad, esto se penaliza.

Tabla 5.13. Ejecución PI-01: Comparador de documentos: Comparación precisa

PI-02	
Módulo :	Algoritmos
Métodos :	*getSimilarity()* de la clase Algorithms
Parámetros :	$n = 3 \quad algorithm = '' kondrak''$
Resultado obtenido :	Se obtiene un resultado de similitud de 0,91. Se obtiene un resultado mejor que en la prueba anterior porque la comparación serializada incluye todos los caracteres del JSON, y como comparten la misma estructura aunque el valor de los atributos difiera mucho, este método ya otorgará un valor alto de similitud.

Tabla 5.14. Ejecución PI-02: Comparador de documentos: Comparación serializada

5.1.3. Pruebas de Sistema

Destacamos que para todas las pruebas realizadas en este punto se ha utilizado el comparador de documentos y no la comparación serializada.

PS-01	
Módulo :	Operaciones difusas
Métodos :	*insertOneFuzzy()* de la clase index
Parámetros :	$threshold = 0{,}7 \quad n = 3 \quad k = 2$
Resultado obtenido :	La función devuelve 0 porque el documento no se ha insertado

Tabla 5.15. Ejecución PS-01: Inserción difusa fallida

PS-02	
Título :	Inserción difusa correcta
Descripción :	Se intentará insertar el documento utilizado para la comparación en el apartado anterior. Se establece un umbral del 0.8
Módulo :	Operaciones difusas
Métodos :	*insertOneFuzzy()* de la clase index
Parámetros :	$threshold = 0{,}8 \quad n = 3 \quad k = 2$
Resultado obtenido :	La función devuelve 1 porque el documento se ha insertado

Tabla 5.16. PS-02: Inserción difusa correcta

PS-03	
Módulo :	Operaciones difusas
Métodos :	*insertManyFuzzy()* de la clase index
Parámetros :	$threshold = 0{,}7 \quad n = 3 \quad k = 2$
Resultado obtenido :	La función devuelve 1 porque ha insertado un documento de dos

Tabla 5.17. Ejecución PS-03: Inserción difusa múltiple

PS-04	
Módulo :	Operaciones difusas
Métodos :	*cleanCollection()* de la clase index
Parámetros :	$threshold = 0{,}7 \quad n = 3 \quad k = 2$
Resultado obtenido :	La función devuelve 1 indicando que ha borrado un documento que resulta ser el que se insertó.

Tabla 5.18. Ejecución PS-04: Limpieza de la colección

PS-05	
Módulo :	Operaciones difusas
Métodos :	*findSimilarDocuments()* de la clase index
Parámetros :	$threshold = 0{,}7 \quad n = 3 \quad k = 2$
Resultado obtenido :	La función devuelve el documento de “Dennis Ritchie”.

Tabla 5.19. Ejecución PS-05: Recuperación difusa

5.2. Conclusiones de la evaluación

Para finalizar el desarrollo del sistema de expansión, comentamos que los resultados que se obtuvieron fueron los esperados.

En primer lugar mencionamos las diferencias entre la comparación serializada y la comparación precisa. La razón de que la primera asigne un valor de similitud mayor que la otra es debido a que compara toda la estructura del documento. Esto produce que también participen en la comparación cosas como el nombre de los atributos, los símbolos de separación (llaves, corchetes, comas...) dando lugar a que si comparamos dos documentos, que aunque la información que contengan sea totalmente distinta, compartan una estructura muy similar, el valor que calculará la comparación serializada será un valor más alto de lo esperado puesto que también compara todas estas cosas. Sin embargo con la comparación precisa se obtiene un valor de similitud más realista puesto que solamente compara el contenido y no la estructura.

Cabe destacar que bien que la solución adoptada para la comparación de los tipos de datos de arrays quizá no arroja resultados muy precisos, pero fue la única que podíamos implementar de manera que el resultado fuese aceptable. De esta forma la comparación precisa de documentos que tengan arrays puede verse afectada arrojando un valor menos preciso de lo que debería.

Por lo demás, los resultados de las comparaciones con los algoritmos son aceptables y las esperadas para cada tipo de algoritmo, y la implementación de estos en conjunto con el comparador de documentos también aporta buenos resultados. Por tanto concluímos que el resultado de las pruebas ha sido negativo puesto que ninguna de ellas ha puesto de manifiesto algún funcionamiento erróneo del sistema.

Finalmente, mencionamos que tal y como se especificó en el capítulo del análisis, el paso final que se realizó en este proyecto fue el despliegue del sistema en el ecosistema de paquetes npm. Para ello se configuró el paquete siguiendo las pautas indicadas en la documentación de npm [7].

[7] `https://docs.npmjs.com/getting-started/publishing-npm-packages`

5.3. Resumen de Ejecución del proyecto y costes

En este punto se presenta de forma detallada la planificación real que se ha realizado durante el ciclo de vida del proyecto, comparándola con la que estimamos y analizando las desviaciones y diferencias que puedan existir entre ambas. También se muestra el cálculo del coste total de forma desglosada para la realización del proyecto, y las posibles desviaciones de coste producidas por las diferencias entre la planificación estimada y real.

5.3.1. Planificación Final

El objetivo de esta sección no es otro que exponer detalladamente el tiempo que se ha dedicado a cada fase del proyecto así como la ejecución temporal general. Esta ejecución queda reflejada mediante un diagrama de Gantt en el cual podemos observar la duración de cada fase y las subtareas que la componen.

El proyecto que nos ocupa comenzó el día 13 de Noviembre de 2017 y finalizó el 14 de Junio de 2018, componiendo un total de siete meses y 1 día de trabajo. Para llevar a cabo la planificación de estos meses, como ya hemos expuesto anteriormente se ha organizado el trabajo principalmente atendiendo a tres fases: la fase en la que se estudian e implementan los algoritmos, el desarrollo de un comparador de documentos y finalmente la integración funcional con MongoDB.

Cuando se comenzó el proyecto ya eramos conocedores de que la duración del mismo debía ser aproximadamente de unos siete meses para cumplir los plazos de entrega sujetos a la convocatoria de Junio tal y como se ha expuesto en el punto *"Planificación"*. No obstante, dados diversos problemas que han surgido a lo largo del ciclo de vida del proyecto, finalmente las cargas temporales de cada fase no han resultado ser exactamente así, aunque si se ha logrado permanecer en el ámbito de siete meses de duración en la perspectiva general.

La primera fase tiene cabida durante aproximadamente los dos primeros meses de trabajo, concretamente desde el 13 de Noviembre de 2017 (día que comienza el proyecto), en el que se da comienzo al estudio del problema, hasta el día 4 de Enero de 2018. En esta fase se empeñan 3 días más del tiempo que nos habíamos propuesto, ya que la fecha estimada de finalización la situábamos el día 1 de Enero de 2018. En estos dos meses aproximadamente, realizamos el estudio de los distintos tipos de algoritmos difusos que se pueden aplicar a nuestro proyecto, además de su implementación y pruebas.

La segunda fase se desarrolla desde la finalización de la primera, el 5 de Enero de 2018, hasta aproximadamente 3 meses después, el 25 de Marzo de 2018. Cabe destacar que en un primer momento y como ya se ha especificado anteriormente, no se pretendía que esta fase tuviese una duración tan larga. Los motivos de esta desviación en el tiempo atienden principalmente a razones técnicas ya que esta ha sido la fase que más análisis y desarrollo ha requerido al tratarse de la más compleja. Debido a esto la planificación (en

la que ya llevábamos un leve retraso de 3 días) se ve muy afectada esta vez, puesto que la fecha estimada de finalización de esta fase se encontraba situada para el día 20 de Febrero de 2018, durando entonces 33 días más de lo estimado.

La última fase referente al desempeño de la funcionalidad del proyecto se desarrolla desde el 26 de Marzo de 2018 hasta el 2 de Mayo de 2018. Al igual que nos ocurrió con la segunda fase, la estimación que se propuso al principio no se corresponde con el tiempo que finalmente se empleó debido a la gran desviación que llevábamos de la fase anterior. Afortunadamente, en este caso, no fue necesario utilizar el tiempo previsto para desarrollar la fase al completo sino que nos llevó menos tiempo completarla. Las razones de que esto ocurriese fue la necesidad de adelantar trabajo más rápidamente debido a que el tiempo empleado en la fase anterior fue mayor de lo que se había previsto, y que las tareas que debíamos realizar en este punto no nos llevaron tanto tiempo al ser meramente de integración y pruebas. Con esto la fase estimada de finalización estaba el 11 de Abril de 2018 y se finalizó el 2 de Mayo, recortando a 21 días la desviación de tiempo.

De forma paralela a estas fases, se redactaban las partes pertinentes de este documento, y finalmente tal y como se había previsto, se completó la redacción de este documento así como la realización del despliegue del sistema desde el 3 de Mayo de 2018 hasta el 14 de Junio de 2018, fecha en la que finaliza el ciclo de vida del proyecto. En este caso, la fecha estimada de finalización se encontraba el día 31 de Mayo de 2018. Tampoco logramos cumplir los objetivos en esta ocasión ya que finalizamos el proyecto con una desviación de 14 días, pero gracias al colchón de tiempo de 14 días que estimamos pudimos completar el trabajo en el las fechas previstas.

Para ver esta información de manera gráfica se proporciona el siguiente gráfico, en el que se puede ver la desviación que han sufrido en tiempo cada una de las fases.

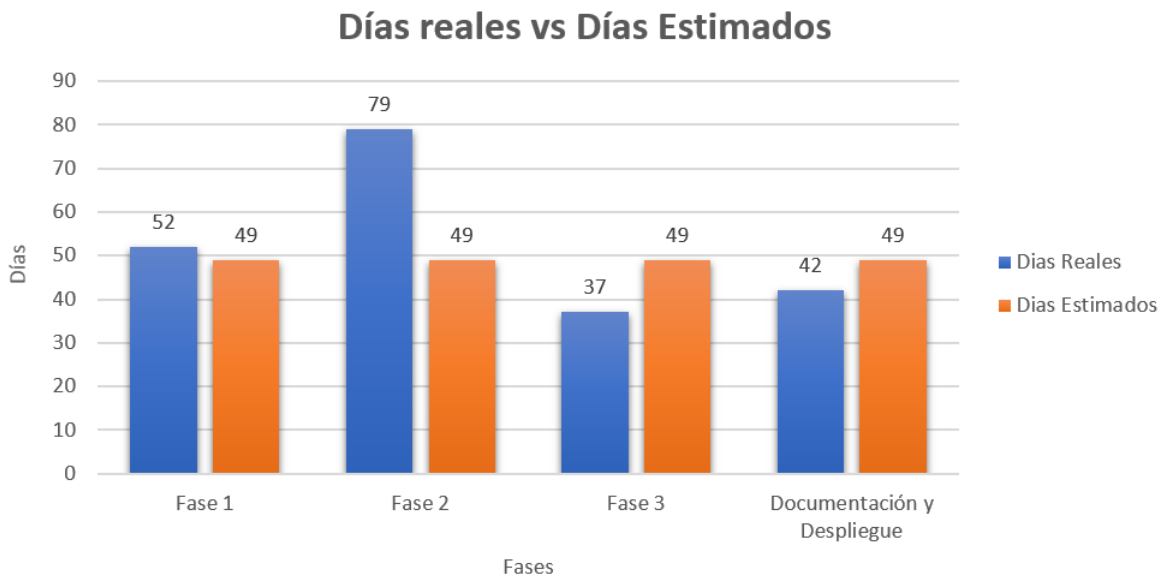

Fig. 5.1. Gráfico comparativo de los días estimados

Finalmente se expone el diagrama de Gantt con la ejecución de la planificación desglosada indicando las fechas en las que se finalizó cada fase.

id	Tarea	Duración	Comienzo	Final
1	Realización completa del trabajo fin de grado	213 días	13.11.17	14.06.18
2	Primera iteración : Algoritmos difusos	52	13.11.17	04.01.18
3	Estudio teórico de los algoritmos	15	13.11.17	28.11.17
4	Análisis y evaluación previa a la implementación	13	28.11.17	11.12.17
5	Implementación de los algoritmos y de la clase Utils	22	11.12.17	02.01.18
6	Planteamiento de la siguiente iteración	2	02.01.18	04.01.18
7	Segunda iteración : Comparador de documentos	79	05.01.18	25.03.18
8	Estudio teórico de la comparación de los documentos	17	05.01.18	22.01.18
9	Análisis y evaluación previa a la implementación	11	22.01.18	02.02.18
10	Implementación del comparador de documentos	44	02.02.18	18.03.18
11	Planteamiento de la siguiente iteración	7	18.03.18	25.03.18
12	Tercera iteración : Implantación en MongoDB	37	26.03.18	02.05.18
13	Estudio teórico de la comparación de los documentos	5	26.03.18	31.03.18
14	Análisis y evaluación previa a la implementación	8	31.03.18	08.04.18
15	Implementación del comparador de documentos	22	08.04.18	30.04.18
16	Planteamiento de la siguiente iteración	2	30.04.18	02.05.18
17	Documentación y despliegue	42	03.05.18	14.06.18
18	Documentación	42	03.05.18	14.06.18
19	Configuración del módulo npm	18	21.05.18	08.06.18
20	Despliegue del módulo en npm	5	09.06.18	14.06.18

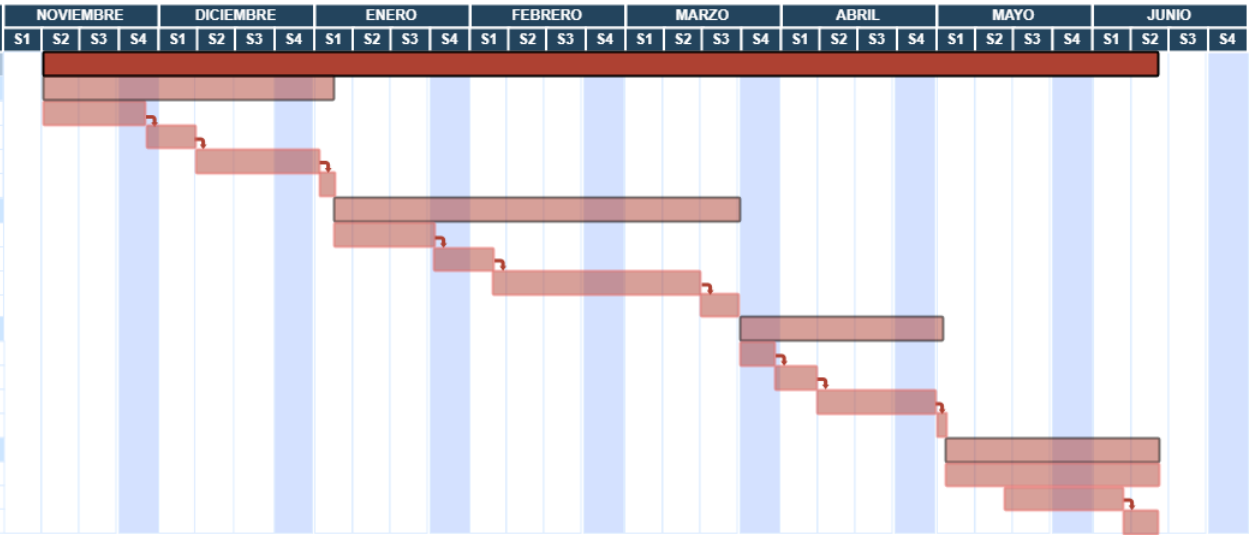

Fig. 5.2. Diagrama de Gantt del proyecto

Una vez mostrado el diagrama de Gantt es importante aclarar información sobre cómo se ha repartido el trabajo durante los 213 días de duración del proyecto.

Debido a la necesidad de compaginar la realización de este proyecto con el trabajo, entre semana (de Lunes a Jueves) a penas se le podía dedicar un tiempo medio de 3 horas en esta franja. En cuanto a los fines de semana, se disponía de más tiempo, y por tanto se empleaba una media de tiempo de 6 horas y media entre el Viernes y el Domingo. Esto hace una media de horas semanales de :

$$Horas_{diasLaborables} + Horas_{diasFinDeSemana} = 3 + 6{,}5 = 9{,}5 \quad horas/semana$$

Puesto que el desempeño en nuestro proyecto se mide en días y no en semanas (a excepción de como se muestran los gráficos del Gantt) debemos calcular el tiempo de trabajo medio al día:

$$\frac{Horas_{semana}}{7} = \frac{9{,}5}{7} = 1{,}36 \quad horas/dia$$

Una vez conocido este dato y sabiendo que el proyecto ha tenido una duración total de 213 días, concluimos que el tiempo total (en horas) de la duración del proyecto ha sido:

$$Horas_{dia} \cdot Dias_{proyecto} = 1{,}36 \cdot 213 = 289{,}68 \quad horas$$

Comentando de forma general las diferencias entre la ejecución real y la estimación que se realizó del proyecto, principalmente, la desviación atiende a dos factores. El primero, la mala estimación que se realizó de las fases, ya que se pensó en un primer momento que todas iban a tener igual duración debido a que no se conocían los aspectos más técnicos de la realización de cada una de ellas. No obstante para la primera y segunda fase la desviación no fue tan grande como ocurrió con la segunda. Esto principalmente fue por la dificultad técnica que requería la realización del algoritmo de comparación, y que finalmente, nos llevó más tiempo lograr que funcionase correctamente.

El segundo factor no es otro que las horas de trabajo al día. Se estimó que se iba a trabajar una media diaria de 1,42 horas al día y luego esta estimación no se cumplió. Si bien es cierto que entre semana (de Lunes a Jueves) no se lograba ni mucho menos dicha media, durante los fines de semana se aumentaba para equilibrar el trabajo que debía realizarse. No obstante, aun así la media de trabajo al día se quedó ligeramente por debajo de lo estimado, en 1,36 horas al día. A pesar de tratarse de una ligera diferencia de tiempo, si la aplicamos de forma acumulada a todo el ciclo de vida del proyecto, nos encontramos con un retraso paulatino del mismo.

Estos dos factores por tanto, han sido determinantes para que se haya producido la desviación antes mencionada, pero afortunadamente estimamos 14 días de margen temporal precisamente para casos como estos, por lo que pudimos completar el trabajo a tiempo.

5.4. Análisis de costes

En esta sección se presenta el cálculo de los costes que ha originado el proyecto que nos ocupa y se tendrá en cuenta las horas determinadas en la sección anterior para ofrecer un desglose final del mismo. Para hacer esto, y al igual que en la sección del presupuesto, se ha estudiado el origen de los costes directos e indirectos.

Los costes directos, compuestos en primer lugar por los costes de personal. Puesto que ya conocemos el coste de un trabajador a euro por hora simplemente multiplicamos este valor por las horas reales que se han dedicado a la ejecución del proyecto. De esta manera obtenemos:

$$Precio_{horas} \cdot Total_{horas} = 16{,}64 \cdot 289{,}68 = 4820{,}28 \quad euros$$

Para los costes de material, al igual que en el presupuesto, no se ha tenido en cuenta el gasto software, sino que se ha tenido en cuenta el gasto hardware. El valor de este ya se especificó en la tabla 3.49, pero al igual que ocurría cuando realizamos el presupuesto, no se debe imputar estos valores de forma íntegra, sino que debemos calcular su coste imputable. En este caso, el tiempo de uso de cada uno de los equipos ha sido de 213 días (duración total del proyecto) por lo que los costes imputables reales son los siguientes:

Concepto	Coste (€)	% Uso	Días Uso	Vida útil (días) [1]	Coste imputable [2]
iMac	1.190	100	213	1825	138,88
Asus	213	100	213	1825	24,86
Ratón	34,99	100	213	1825	4,08
TOTAL	**1.437,99**				**167,82**

[1] El valor de la vida útil de un elemento de hardware se establece en 5 años.

[2] El coste imputable se calcula utilizando la amortización $(Tiempo_{usoEnProyecto}/Tiempo_{vidaUtil}) \cdot Coste_{equipo} \cdot \%Uso$

Tabla 5.20. Costes reales imputables del hardware

Por tanto sumando esta cifra de la maquinaria amortizada y los costes de personal, obtenemos que los costes directos son los siguientes:

$$Costes_{Personal} + Costes_{Materiales} = 4820{,}28 + 167{,}82 = 4988{,}1 \quad euros$$

Aplicando de igual manera que en el presupuesto, unos costes indirectos del 20 %, y aplicando un IVA del 21 %, obtenemos el siguiente desglose de costes:

Concepto	Valor (€)
Costes directos	4988,1
Costes indirectos	997,62
Costes totales Sin IVA	5985,72
Costes totales Con IVA	7242,72
TOTAL	**7242,72**

Tabla 5.21. Desglose del coste real total

Por tanto el coste final que se ha empeñado en la realización de este proyecto ha sido de **SIETE MIL DOSCIENTOS CUARENTA Y DOS COMA SETENTA Y DOS** euros.

Si realizamos la operación de la diferencia entre el presupuesto inicial y el coste real del proyecto podemos observar lo siguiente:

$$Coste_{Presupuestado} - Coste_{Real} = 7498,08 - 7242,72 = 255,36 \quad euros$$

Lo cual nos deja con el valor positivo de 255,36 €que se han presupuestado de más.

Este valor proviene de los cálculos de costes que se hicieron para el coste de personal para los días de colchón y los costes de material imputable. Dichos valores, que ascendían a 332,8€y 11,13€respectivamente, suponían un margen económico en el caso de que el proyecto se alargara más de lo debido en el tiempo y hubiese que destinar más recursos. Por tanto, si aplicamos el IVA (ya que en el presupuesto se aplicaba sobre el montante del coste total) a estos dos valores, encontramos que disponemos del siguiente margen económico sobre el precio total del proyecto.

$$(Costes_{PersonalAdicional} + Costes_{MaterialAdicional}) \cdot 1,21 = 416,15 \quad euros$$

Puesto que la diferencia entre la estimación y el coste real es un valor positivo, esto indica que estos 416,15€que se sobreestimaron por si acaso, no se han agotado en su totalidad como vemos en la imagen.

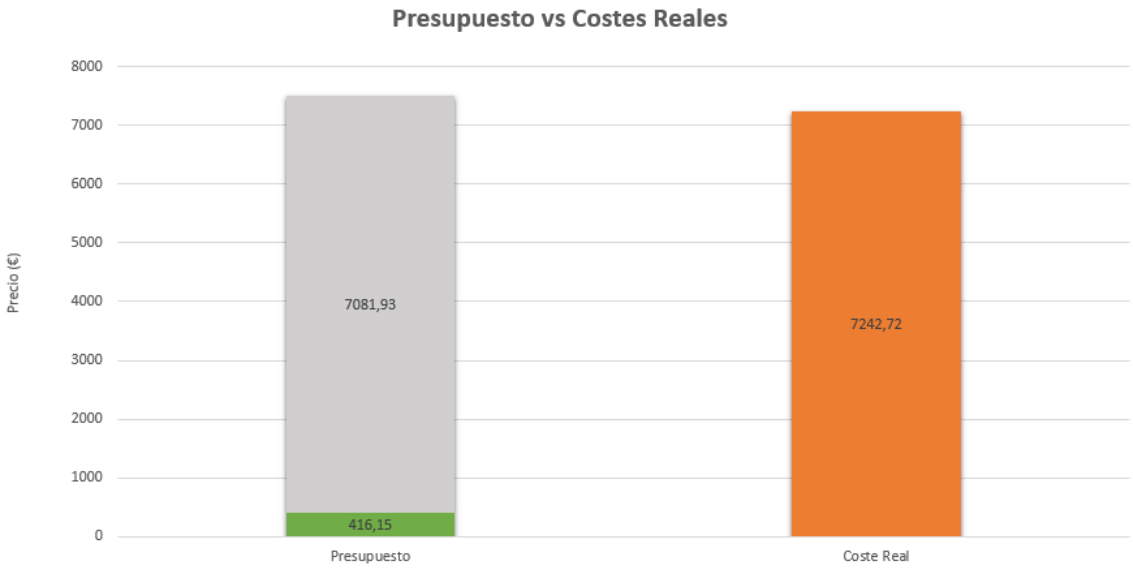

Fig. 5.3. Costes reales vs estimados

La barra naranja que representa el coste real del proyecto se queda ligeramente por debajo de la otra. Esto es principalmente debido gracias a que, como se estimó un presupuesto mayor que el utilizado, el cómputo total de nuestro proyecto resulta se menor. Podemos preguntarnos como es esto posible si en la planificación temporal ya mencionábamos que nos comemos todo el tiempo de colchón que se había establecido. La respuesta es sencilla: Finalmente no se ha trabajado tanto como se estimó debido principalmente a que se dedicaban menos horas de las estimadas al día. Este factor, como ya se ha explicado, nos llevó a utilizar el tiempo de margen en su totalidad, puesto que si no, no seríamos capaces de entregar el proyecto a tiempo. Sin embargo para el caso de los costes, estos se reducen al no haber trabajado tantas horas como se estimó en un principio. Sin embargo, en el caso de los costes imputables estos si que son mayores ya que este factor depende directamente del tiempo que se dedique al proyecto y aquí si tenemos más desviación.

Por tanto concluimos que la estimación de los costes a pesar de la desviación en el tiempo, ha sido óptima, ya que no se ha rebasado la cifra de presupuesto que se estableció inicialmente y tampoco hemos sobreestimado el coste del proyecto.

6. CONCLUSIONES

En este capítulo se exponen las conclusiones obtenidas tras la realización del trabajo realizado. En primer lugar se identifican los objetivos que se han cumplido en función de los que nos propusimos en un primer momento al inicio del trabajo. En esta sección se discute la satisfabilidad del sistema final y si este cumple dichos objetivos iniciales, así como las desviaciones que puedan haberse sucedido respecto a los objetivos inicialmente propuestos.

Seguidamente se mencionan aplicaciones nuevas que pueden implementarse al sistema desarrollado para completarlo así como nuevas líneas de investigación que se abren tras haber realizado el trabajo que nos ocupa.

6.1. Objetivos cumplidos

Como expusimos en la introducción de este trabajo, el proyecto constaba de un único objetivo: la ampliación de las operaciones que ofrece un sistema gestor de base de datos NoSQL por defecto.

Una vez que se ha finalizado el proyecto y miramos con perspectiva la solución expuesta, concluimos que el objetivo que se perseguía se ha cumplido de manera satisfactoria. En un principio, cuando nos propusimos la realización de este trabajo teníamos en mente la adición de operaciones sencillas de manera que así proporcionásemos una base sólida sobre la que poder construir nuevas herramientas en un futuro. Finalmente, desarrollamos mucho más de lo que nos habíamos propuesto en un principio puesto que hemos sido capaces incluso de establecer un algoritmo que sea capaz de comparar documentos completos de forma independiente a la estructura que posea el mismo.

Por tanto, y como se vaticinaba en la introducción de este documento, el sistema final que se ha realizado aquí es solamente una pequeña pieza de algo que puede llegar a ser mucho mayor. De esta casuística precisamente es por la que decidimos comenzar por algo tan básico como las operaciones CRUD a la hora de implementar lógica difusa en una base de datos NoSQL.

Esto ligado a los resultados positivos que ha arrojado la realización de este trabajo nos dejan con una perspectiva halagüeña en la que se pueden continuar las vías que se han ido construyendo a lo largo de este proyecto y ampliar su alcance a dimensiones aun mayores, tal y como mencionamos en el siguiente punto.

6.2. Líneas futuras de trabajo

Para cerrar finalmente este trabajo, debemos decir que quedan abiertas con su realización muchas líneas de investigación. Estas las podemos dividir principalmente en dos ramas: a corto y largo plazo.

Aquellas que quedan bajo la calificación de "corto plazo" son principalmente mejoras para el sistema que se ha desarrollado. Entre estas líneas futuras destacamos sobre todo las siguientes:

- El desarrollo de un algoritmo más óptimo que los que se han presentado aquí a la hora de realizar la comparación de los documentos, puesto que al tratarse de un algoritmo recursivo, tiene un alto coste en tiempo de ejecución.
- A la hora de inspeccionar las colecciones para realizar las operaciones difusas implementadas, en lugar de recorrer estas de forma secuencial, recorrerlas de forma paralela de manera que se inspeccionen un mayor número de documentos en una menor cantidad de tiempo (utilizando la tecnología de Sharding que proporciona MongoDB). Así de esta manera podríamos aplicar estos métodos a colecciones de mayor tamaño, que al final es una característica necesaria si pretendemos desarrollar herramientas que hagan uso de la lógica difusa para tratar con la gran cantidad de datos que pueden almacenar las bases de datos NoSQL.
- Idear un algoritmo para la comparación entre arrays que sea más preciso a la hora de dar un valor de similitud.
- Continuar enriqueciendo la extensión funcional de carácter básico para que la solución vaya creciendo, como por ejemplo técnicas difusas aplicadas a las operaciones de *pipeline* de MongoDB (en el caso que nos ocupa que ha sido desarrollar las operaciones difusas para este software concreto).
- Implementar a la solución algoritmos nuevos para el cálculo de la similitud. Extender dicho cálculo, no solamente restringiéndonos al uso de la aproximación de cadenas (que es el caso que hemos tratado a lo largo del proyecto) sino implementar algoritmos que permitan el establecimiento de la similitud mediante técnicas semánticas y de equivalencia.

En cuanto a las líneas futuras que reciben el calificativo de "largo plazo" con estas nos referimos a aplicaciones de mayor envergadura que las operaciones que se han implementado en este trabajo. Como ya hemos mencionado, disponer de variedad de herramientas difusas (más allá de las aquí desarrolladas) para las bases de datos NoSQL nos permitiría poder tratar la información no perfecta, y descubrir nuevas aplicaciones:

- Podríamos utilizar ese futurible set de herramientas en el ámbito de la ciber-seguridad: para detectar comportamientos sospechosos, fraudes y ataques aprovechándonos

del factor de velocidad que ofrecen las bases de datos NoSQL y de las herramientas difusas desarrolladas que permitirían establecer patrones sobre los ataques y de esta forma detectarlos casi al instante.

- Enriquecer las aplicaciones orientadas a la semántica, ya que, aunque en este trabajo no se han implementado técnicas difusas relacionadas con ella, podrían añadirse y utilizarse para establecer relaciones semánticas de manera automática en los documentos almacenados en una colección, lo cual nos ayudaría en el trabajo del descubrimiento del conocimiento y formación de ontologías.

No obstante, previamente a poder pensar en realizar este tipo de aplicaciones, debemos pensar en seguir fortaleciendo las operaciones aquí propuestas y aumentar las posibilidades que el sistema que hemos desarrollado ofrece. De esta forma, trabajando en la implementación de técnicas difusas en las bases de datos NoSQL, finalmente alcanzaremos la codificación de un conjunto de potentes herramientas que nos servirán para analizar todos esos datos que hoy en día conforman nuestra sociedad.

7. ENGLISH SUMMARY

In this chapter we will accomplish a summary of the most important points that the work “Functional Extension of NoSQL DBMS by applicarion of diffuse techniques” includes.

Firstly we will talk about the motivation that have led us to perform the proyect, and the objectives associated with it. Secondly, we will expose in a summarized way, how we have approached the problem and the final solution that has been adopted. Finally we mention the conclusions we have obtained from the realization of the work and the future lines of work that can be adopted through the use of our project.

7.1. Motivation and Objectives

Nowadays we live in an interconnected society, in which, in one way or another, the amount of information we have does nothing but grow. The rise of social networks, forums, mobile applications, web applications, etc. has generated a large amount of data of different natures.

In statistics provided by the companies ’We Are Social’ and ’Hootsuite’ [1] of a study conducted in January this year, the population of devices connected to the Internet is 7.593 billion. This data is surprising because the world population in January 2018 was 7.576 billion. How can there be more connected devices than the number of people in the world? The answer is simple: we are in the era of computerization and today many devices connect to the internet to work properly. In addition, it is very usual for a single person to have more than one device that connects to the network. These two things generate the phenomenon described that the number of devices connected to the network exceeds (and continues to exceed in the future) the total world population. Therefore, this situation produces the increase of data that we talked about previously, and we must think about how to store them.

The nature of this type of analytical processing relaxes these requirements, since, as dictated by the “Law of large numbers” **numbers** we can afford to dispense with a certain part of the data without affecting the result, at least, in a statistically significant way. We can take advantage of this situation to apply a distributed approach in which the relaxation of the exhaustivity parameter can compensate the requirements of the CAP theorem: it is impossible to reach a distributed system while maintaining availability, consistency and fault tolerance. Therefore, the distribution is considered as a plausible solution to the massive storage required, and parallel processing as an efficient solution for the analysis of such a size of data. The technology that has filled this space has been called NoSQL (Not Only SQL) and will be named in this way from now on throughout the document.

We can answer this question in the following way; if we are able to analyze the data produced by all the devices connected to the internet we can extract valuable information from this data and apply it to different environments. For example, the financial environment, the ability to personalize (offer each user information of interest to him) and even marketing. In this way, in any area that we imagine we will have the capacity to improve it thanks to an analysis of the data that directly affect it.

We will not go into details of how the analysis of so many data is performed, since it exceeds what we want to expose at this point. It is enough to say that it is the tools provided by the NoSQL databases that allow this analysis to be possible (together, of course, of professionals who know how to use it).But normally, the data that we have to analyze is not accurate information, since it is often vague information. In other words: for human knowledge it is easily understandable, but not for a computer, and we cannot program a machine to analyze data if it does not understand the nature of them. This is why other technologies that are capable of analyzing this type of inaccurate information take on importance and are called fuzzy techniques.

Although it seems surprising, there are no (as we see throughout the document) many technological models that contemplate the widespread use of these techniques to treat information. Commonly, we use the operations and tools provided by the database managers in a native way and we leave aside the option of analyzing the data with fuzzy techniques with which, perhaps, we would be able to obtain a deeper analysis of the data. That is why this work is motivated to unite both facets, NoSQL databases and fuzzy techniques to cover this situation and see if it is possible to use these techniques to expand the options offered by a current NoSQL manager.

That said, we set ourselves a single goal in this project; the addition of new operations to a NoSQL database manager through the use of these techniques as a first approach to the use of fuzzy techniques in this field. This objective is far from generating a final product that can be applied directly to the NoSQL databases and provides a multitude of diffuse tools. Mainly because carrying out a complete study of the technology by applying it satisfactorily to the analysis of a large amount of data exceeds the purpose of the final year project, which is the reason why we carry out this work.

7.2. Used technology

To fulfill the objectives that we have proposed, we must bear in mind that there are several types of NoSQL database managers. In the first place, there are several paradigms and each of them is characterized by the use that is given to it. Without going into detail, since this study is carried out more extensively in chapter 3, the existing paradigms are the following: key-value, document oriented, graph oriented and column oriented.

We have chosen the document-oriented paradigm to fulfill the work, because this technology is the most appropriate for the task we want to perform. Nowadays it is the most used of all paradigms of existing NoSQL databases, it is also the most versatile and has a simple operation thanks to which it stores the information following a standard format. This last feature helps us when implementing diffuse operations, and the fact that it is such an expanded technology can lead to a greater impact of our work.

Once the document-oriented paradigm has been chosen, we have to choose the software in which the solution will eventually be developed. For this we compared two software that implement the document-oriented databases and both are quite widespread: MongoDB and CouchDB. The comparison of both technologies has been made in the section *"Viabilidad del sistema"*, but, briefly, we have chosen MongoDB because it has more advantages than CouchDB; First of all we had worked with MongoDB before, which makes the task easier. In addition, the query language of MongoDB is JavaScript, with which we had also worked, while CouchDB has a query language that works through a REST API, with which we had never worked before. These two aspects make MongoDB as the final choice. Besides, this is more extended than CouchDB, so we can expect the work to have a better reception.

Therefore, the MongoDB technology (a document-oriented NoSQL database) has finally been chosen to carry out the objectives that we have proposed in this work. Once we have chosen the software, we must choose how to implement the solution. For this we propose three implementation options: JavaScript + MongoDB, Node.js + MongoDB and Node.js + Compass.

The first uses the JavaScript language that is capable of running on the MongoDB console itself. However, we discarded it quickly because we need the new functions to be available, regardless of the connection type with MongoDB or where it is installed. With this implementation we have to develop the solution, either in a single file that MongoDB loads every time it creates an instance (thus having the functions available), or inserting it into a special table that is used to store JavaScript functions. Both solutions have the disadvantage that it is not very scalable and also makes installation difficult if we do not have administrator access for MongoDB.

In the second case we use the Node.js framework to make the connection and the operations with MongoDB. This case is the one that most resembles the objective that we have proposed, since the solution can be codified in the form of an API. In addition,

we continue to maintain the JavaScript language as the language in which we will develop, and eliminate the disadvantages of the previous case. In this case Node.js has a huge package ecosystem called npm, in which solutions can be deployed for other users to use. Specifically, to make the connection with the MongoDB database we can use a package (officially developed by the MongoDB team) and so eliminating the installation disadvantage mentioned above. We can also deploy our system in npm and make it easily accessible. We also eliminate the disadvantage that it is not scalable since in developing the system in this way we can develop it in different files, establishing a hierarchy of classes and methods that is easier to expand and maintain afterwards.

In the third case, we have the option to use Node.js (in the same way as the previous case) but also adding a graphical interface called MongoDB Compass. This interface is provided by the MongoDB team itself and allows modules to be added to it through React.js. The advantage of making this implementation is that we would have a robust final product since the new functionalities would be incorporated into an interface. However, this development exceeds the purpose that we had proposed for this work, since it needs much more dedication, although it would be a good choice for future work.

For these reasons, we have finally chosen the second case: Node.js + MongoDB to carry out our solution. However, in this section we have explained in a very summarized way the reasons why we have made each of the elections. For a more detailed explanation, we recommend reading chapter three written in Spanish, in which these aspects are explained as well as others that have not been included in this chapter; for instance, the definition of requirements or the planning of the project.

7.3. System operation

At this point we detail the design of the system architecture that we want to implement. First of all we emphasize that it is a summary, the complete content of the design is found in chapter *"Descripción modular"*. Therefore, in this chapter we present a general description of the system we have implemented.

First of all, to say that we have made the decision to implement fuzzy functions that are related to the CRUD operations that MongoDB allows. First of all, to say that we have made the decision to implement fuzzy functions that are related to the CRUD operations that MongoDB allows. Therefore, the functions of fuzzy erasure, fuzzy insertion and fuzzy document retrieval are intended to be implemented. We have left aside the fuzzy update function because if we update all the documents that are similar then we would increase the similarity between them, and even in the worst case we would have repeated information.

Therefore, to develop these functions, we must first be clear about the theory we are going to use: very briefly, there are algorithms that are able to measure the similarity between two strings of characters and other data types. These algorithms establish a degree of similarity in the interval $[0, 1]$. Therefore when the result is close to 1, the elements compared will be very similar, while if it is close to 0, they will be different. The fundamentals of these algorithms are explained in chapter two, where we expose the fuzzy logic proposed by Zadeh and the approximate string matching algorithms. To perform the fuzzy operations that we have proposed, we must be able to compare the MongoDB storage unit; the documents, for which we will make use of these algorithms. Therefore, by comparing the documents and studying their similarity value, we will erase them, insert or recover them, as we will explain later.

Therefore, the expansion system (as we will call it from now on) is divided into three large parts, and each one of them has been developed in a different phase (as we can see in the planning of the project). The first one consists of achieving the implementation of the diffuse algorithms that we have mentioned. The second consists of the development of the component that is capable of comparing complete documents, relying on the algorithms implemented in the first phase and giving a similarity value between them independently of the types of data that compose them and their structure. Finally, the last part consists of implementing the functions of insertion, recovery and deletion using the document comparator.

It should be noted that the spiral methodology was followed to carry out this work since it was the one that best adapted to the objectives we had proposed. This methodology allows to separate the work that we must carry out in phases, four specifically:

- Definition of objectives: in this phase of the project we define the objectives that we intend to achieve in this phase. We analyze the situation in addition to identifying the risks and restrictions that we can find and, due to this, the approach of alternative

strategies.

- Risk assessment: In this phase we evaluate the alternatives that we have identified due to the risks and define the steps to follow.
- Development and Testing: After evaluating the alternatives, we chose one of them to carry out the development of the system. After the development we perform tests on it to check and ensure that the system works correctly.
- Planning of the next phase: In this phase we made a review of the progress of the project so far and we decided to continue or not with another cycle of the spiral. In the case of making the decision to continue, we will start again with the first point of this list, if we do not continue, the project's life cycle will have ended and we will have reached the end of the spiral.

Therefore during the duration of the project we have carried out these four steps for each cycle of the spiral. These cycles have been defined as the phases that we have described previously. Next, we explain how we have developed each one of them.

First of all, we must mention that we have integrated these modules in the form of an API. As we will see in the next point, the final form that the implemented methods has taken is very similar to the one implemented by the Node.js module for MongoDB that has been used to make the connection. However, the functionality performed by the expansion system must be transparent to the user, and be compatible regardless of the nature of the connection with MongoDB (Local, cluster ...), the database and the collection of documents, by both the API will have configurable options in which, prior to the use of the expansion system, the MongoDB connection string will be indicated, the name of the database and collection on which the operations are to be carried out. In addition, and as mentioned in the following lines, we have developed another method additional to the comparator module to establish a similarity between two documents. The option to use one or the other is also included in the configurable options of the API. Finally, we also introduce some parameters that the algorithm module requires for its correct operation. The interface of the API, as well as the options available to us, have been explained at point *"Descripción modular"*.

7.3.1. Algorithm module

The algorithm module collects the implementation in several classes of algorithms of chain approximation and fuzzy belonging that are necessary to measure the similarity between different types of data in the MongoDB documents. Specifically, we have implemented the Jaccard algorithm, the kondrak algorithm, the levenshtein algorithm and the diffuse Gauss membership function. The first three are used when we want to compare data that are strings of characters, while the last one is useful when we want to compare numbers or dates.

Therefore, this module will be essential for the correct operation of the expansion system, since the logic that is responsible for giving the similarity that has two entries (whether strings of characters, numbers or dates) is contained here.

We also implemented a method that acts as an interface to the chain approach algorithms. This is responsible for executing the similarity of a document in a different way as we will see in the following point: Serialize the JSON document that is compared (that is, it is completely converted to a String) and directly apply one of these approximation algorithms of chains to establish the similarity between the two. However, this method has the disadvantage that we compare absolutely all the characters of the document (including keys, square brackets, commas ...) which can cause the similarity value to have noise. For example, if we compare two documents that do not have the same structure but that the information they contain is not similar, this method can give a higher similarity value than it should because they share the structure.

7.3.2. Comparator module

The module of the document comparator is undoubtedly an important part of the expansion system, however, the correct operation of the diffuse operations do not depend entirely on it, since they implement another method of making the comparison apart from the use of this module as mentioned in the previous section. For the rest, this module consists of a single class which implements an algorithm that returns a similarity value between two documents that are indicated.

In order to do this, a recursive function is applied that runs through the document and compares the values of each attribute that these documents have in common. If the data type is a character string it will apply the levenshtein algorithm (if it is a short string) or kondrak (if it is a long string), whereas if the data type is a number or date it will apply the Gauss membership function. In the case of arrays, we have implemented the jaccard algorithm in its diffuse form. Therefore, to establish the intersection and union we use a degree of belonging, and when we compare one element with another we check if its similarity is greater than the specified degree to add it to the intersection set or not. Finally, in the case of nested objects, we make a call to the same function that is implemented in the module, generating a recursive algorithm. This is why this method, although it is more realistic than serialization, has a very high cost in time. The larger the document and the more nests you have, the longer it will take to complete the execution.

7.3.3. Fuzzy operations module

The module that implements the fuzzy operations is responsible for implementing the final operations on the database. It will be in charge of connecting to MongoDB through the driver for Node.js and using the similarity between documents to carry out the operations developed:

- In the case of fuzzy insertion, the collection in which it is intended to be inserted is traversed, comparing the document to be inserted with the stored ones. If the calculated similarity of the document that is inserted with any of the stored ones exceeds the marked similarity threshold, the document will not be inserted. However, if none of them exceeds the threshold, if the insertion will proceed. It should be noted that the version of inserting only one document, or inserting several, have been implemented.

- In the case of fuzzy erasure, we have implemented it as a way to "clean" the collections. We go through the entire collection by posting a document and comparing it with others. If any of the comparisons made yield a similarity value above the threshold, then that document will be deleted. When the cleaning has finished for that first document, a new one will be set to start the cleaning cycle again and we will repeat this flow for each document in the collection. In this way, at the end of the execution, the documents that could be repeating information or very similar to those already existing and that did not add value to the stored data set will be eliminated.

- In the case of fuzzy recovery, the collection is searched for those documents that have a similarity with the specified document. If it exceeds the stipulated similarity threshold then that document will be recovered, while if it does not exceed it, we will not recover it. Finally, and as we mentioned above, the comparison is made with complete documents, so if you want to perform a diffuse search of a document, looking only for an attribute, probably the results obtained are not as expected. This will be explained more in depth in the following point but, in broad terms, what happens is that if we only perform the fuzzy search indicating an attribute, when making a comparison of complete documents, the algorithm takes the indicated attribute as a complete document and will penalize (decreasing the similarity factor) when comparing documents that surely have more attributes than the one indicated.

Finally, there is a small module of " utilities " in which two classes are implemented: a first one called " Shingles " which is used to obtain the ngram of a string of characters and a second one called " Spinner " that draws a load bar in the console during the execution of fuzzy operations to offer the user information about the execution status since, although the causes are mentioned later, the more information the DB contains, the more the execution will be delayed.

7.4. Conclusions and Future works

In this chapter the conclusions obtained after the realization of the work carried out are exposed. In the first place, the objectives that have been met are identified according to those we proposed at the beginning of the work. In this section the satisfaction of the final system is discussed and if it meets these initial objectives, as well as the deviations that may have occurred with respect to the initially proposed objectives.

Next, new applications are mentioned that can be implemented to the system developed to complete it, as well as new lines of research that are opened after having carried out the work in question.

7.4.1. Conclusions

As we explained in the introduction to this work, the project consisted of a single objective: the expansion of the operations offered by a NoSQL database management system by default.

Once we have finalized the project and we look with perspective at the exposed solution, we conclude that the objective that was pursued has been fulfilled satisfactorily. In the beginning, when we proposed the realization of this work we had in mind the addition of simple operations to provide a solid base on which to build new tools in the future. Finally, we developed much more than we had originally proposed, since we were even able to establish an algorithm that is capable of comparing complete documents independently of the structure that it has.

However, and as predicted in the introduction of this document, the final system that has been made here is only a small piece of something that can be much greater. This is precisely why we decided to start with something as basic as CRUD operations when implementing fuzzy logic in a NoSQL database.

This is linked to the positive results that this work has produced they leave us with a promising perspective in which we can continue the ways we have been building throughout this project and expand its reach to even greater dimensions, as we mentioned in the following point.

7.4.2. Future works

To close this summary, we must say that many lines of research are open with its realization. These can be divided mainly into two branches: short and long term.

Those that remain under the "short termrating are mainly improvements to the system that has been developed. Among these future lines, we highlight the following:

- The development of a more optimal algorithm than the ones we have presented here

to perform the comparison of the documents, since being a recursive algorithm, it has a high cost in execution time.

- When inspecting the collections to perform the diffuse operations implemented, instead of traversing them sequentially, traverse them in a parallel manner so that a greater number of documents are inspected in a shorter amount of time (using Sharding technology that provides MongoDB). This way we could apply these methods to larger collections, which in the end is a necessary feature if we want to develop tools that make use of fuzzy logic to deal with the large amount of data that NoSQL databases can store.
- To devise an algorithm for the comparison between arrays that is more precise at the time of giving a value of similarity.
- Implement more basic features for the solution to grow, such as fuzzy techniques applied to the operations of *pipeline* of MongoDB (in the case that concerns us that has been to develop the diffuse operations for this particular software).
- implement new algorithms in the solution for the calculation of similarity. Extend this calculation, not only restricting ourselves to the use of the chain approach (which is the case we have dealt with throughout the project), but also implementing algorithms that allow the establishment of similarity using semantic and equivalence techniques.

Regarding the future lines that receive the qualification of " long term " with these we refer to applications of greater scope than the operations that have been implemented in this work. As we have already mentioned, having an ecosystem of diffuse tools (beyond those developed here) for NoSQL databases would allow us to be able to treat non-perfect information, and discover new applications:

- we could use that futuristic set of tools in the field of cyber-security: to detect suspicious behavior, fraud and attacks taking advantage of the speed factor offered by the NoSQL databases and the diffuse tools developed that would allow us to establish patterns over the attacks and in this way detect them almost instantly.
- The applications oriented to semantics, since, although in this work no diffuse techniques related to it have been implemented, they could be added and used to establish semantic relationships automatically in the documents stored in a collection, which would help in the work of discovering knowledge and forming ontologies.

However, before being able to think about making this type of applications, we should think about continuing to strengthen the operations proposed here and increase the possibilities that the system we have developed offers. In this way, working on the implementation of fuzzy techniques in NoSQL databases, we will finally achieve the codification of a set of powerful tools that will help us analyze all those data that today make up our society.

BIBLIOGRAFÍA

[1] W. A. Social y Hootsuite. (ene. de 2018). Digital in 2018, [En línea]. Disponible en: https://www.slideshare.net/wearesocial/digital-in-2018-global-overview-86860338 (Acceso: 28-05-2018).

[2] I. Castelar. (mayo de 2017). Ley de los Grandes Números, [En línea]. Disponible en: https://iescastelar.educarex.es/web/departamentos/matematicas/matematicasccss2ba/matematicas2ccss/ttcentrallimite.htm.

[3] L. Zadeh, "Fuzzy sets", *Information and Control*, vol. 8, n.º 3, pp. 338-353, 1965. [En línea]. Disponible en: http://www.sciencedirect.com/science/article/pii/S001999586590241X.

[4] D. de lenguajes y ciencias de la computación. Universidad de Málaga. (abr. de 2018). Teoría de conjuntos difusos y lógica difusa, [En línea]. Disponible en: http://www.lcc.uma.es/~eva/aic/apuntes/fuzzy.pdf.

[5] M. I. Durán Vicente y T. Benito Matías. (abr. de 2018). Lógica Borrosa, [En línea]. Disponible en: http://www.it.uc3m.es/jvillena/irc/practicas/08-09/10.pdf.

[6] J. M. Medina, O. Pons y M. A. Vila, "Gefred: A generalized model of Fuzzy Relational Databases", *Information Sciences*, vol. 76, n.º 1, pp. 87-109, 1994. DOI: https://doi.org/10.1016/0020-0255(94)90069-8. [En línea]. Disponible en: http://www.sciencedirect.com/science/article/pii/0020025594900698.

[7] D. Sanchis Minguez, "Bases de Datos Relacionales Difusas", Universitat Jaume I, sep. de 2015.

[8] Y. C. Fonseca Reyna, O. G. Reyes Pupo, M. Aballe Rodríguez y A. Urquiza Jiménez, "Una mirada a las bases de datos difusas", *Revista Cubana de Ciencias Informáticas (RCCI)*, vol. 6, n.º 3, sep. de 2012.

[9] M. Gilleland. (feb. de 2018). Levenshtein Distance, in Three Flavors, [En línea]. Disponible en: https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm.

[10] E. Albacete, F. J. Calle-Gómez, E. Castro y D. Cuadra, "Semantic Similarity Measures Applied to an Ontology for Human-Like Interaction", *CoRR*, vol. abs/1401.4603, 2014. arXiv: 1401.4603. [En línea]. Disponible en: http://arxiv.org/abs/1401.4603.

[11] M. Porter. (feb. de 2018). The Porter Stemming Algorithm, [En línea]. Disponible en: https://tartarus.org/martin/PorterStemmer/.

[12] A. Castelltort y A. Laurent, "Fuzzy Queries over NoSQL Graph Databases: Perspectives for Extending the Cypher Language", en *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, A. Laurent, O. Strauss, B. Bouchon-Meunier y R. R. Yager, eds., Cham: Springer International Publishing, 2014, pp. 384-395.

[13] A. Belhadj Kacem y A. Grissa Touzi, "Towards Fuzzy Querying of NoSQL Document-oriented Databases", en *DBKDA 2015, The Seventh International Conference on Advances in Databases, Knowledge, and Data Applications*, mayo de 2015, pp. 153-158.

[14] R. M. Gómez Labrador. (sep. de 2015). TIPOS DE LICENCIAS DE SOFTWARE, [En línea]. Disponible en: `http://www.informatica.us.es/~ramon/articulos/LicenciasSoftware.pdf` (Acceso: 25-05-2018).

[15] F. S. Foundation. (abr. de 2018). El sistema operativo GNU, [En línea]. Disponible en: `https://www.gnu.org/licenses/licenses.es.html` (Acceso: 25-05-2018).

[16] A. CouchDB. (abr. de 2018). Apache CouchDB 2.1 Documentation, [En línea]. Disponible en: `http://docs.couchdb.org/en/2.1.1/intro/index.html`.

[17] cURL. (abr. de 2018). command line tool and library for transferring data with URLs, [En línea]. Disponible en: `https://curl.haxx.se/`.

[18] M. Sarig. (abr. de 2018). CouchDB vs MongoDB, [En línea]. Disponible en: `https://blog.panoply.io/couchdb-vs-mongodb`.

[19] K. B. of Relational y N. D. M. Systems. (abr. de 2018). DB-Engines Ranking - Trend of CouchDB vs. MongoDB Popularity, [En línea]. Disponible en: `https://db-engines.com/en/ranking_trend/system/CouchDB%3BMongoDB` (Acceso: 13-05-2018).

[20] M. Team. (2008). Mongo Files, [En línea]. Disponible en: `https://docs.mongodb.com/manual/reference/program/mongo/#mongo-mongorc-file` (Acceso: 15-05-2018).

[21] ——, (2008). Mongo Store JavaScript Function, [En línea]. Disponible en: `https://docs.mongodb.com/manual/tutorial/store-javascript-function-on-server/` (Acceso: 15-05-2018).

[22] ——, (2008). Creating Compass Plugins, [En línea]. Disponible en: `https://docs.mongodb.com/compass/master/plugins/creating-compass-plugins/` (Acceso: 15-05-2018).

[23] *Pruebas Software*, Español, Online, Facultad de Ingeniería, 2011. [En línea]. Disponible en: `http://materias.fi.uba.ar/7548/PruebasSoftware.pdf` (Acceso: 17-05-2018).

[24] O. Tinoco Gómez, P. P. Rosales López y J. Salas Bacalla, "Criterios de selección de metodologías de desarrollo de software", Español, *Industrial Data*, vol. 13, pp. 70-74, 2010. [En línea]. Disponible en: http://www.redalyc.org/articulo.oa?id=81619984009.

[25] E. M. Méndez Nava, "Modelo de evaluación de metodologías para el desarrollo software", Tesis de mtría., Universidad Católica Andrés Bello, jul. de 2006. [En línea]. Disponible en: www.academia.edu/download/38917027/AAQ7365.pdf.

[26] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", *Computer*, vol. 21, n.º 5, pp. 61-72, mayo de 1988. DOI: 10.1109/2.59. [En línea]. Disponible en: http://dx.doi.org/10.1109/2.59.

[27] I. Sommerville, *Ingeniería del Software*. Pearson Education, 2005, pp. 68-70. [En línea]. Disponible en: https://books.google.es/books?hl=es%5C&lr=%5C&id=gQWd49zSut4C.

[28] U. C. I. de Madrid. (2018). Ficha Reina Trabajo Fin de Grado. Ingeniería Informática. Plan 2011, [En línea]. Disponible en: https://aplicaciones.uc3m.es/cpa/generaFicha?est=218%5C&asig=13895%5C&idioma=1 (Acceso: 18-05-2018).

[29] ——, (2018). Evaluación Continua. Créditos ECTS, [En línea]. Disponible en: https://www.uc3m.es/ss/Satellite/Grado/es/TextoMixta/1371214034625/Evaluacion%5C_continua (Acceso: 18-05-2018).

[30] *Informe Infoempleo 2016*, Adecco Group, 2017. [En línea]. Disponible en: https://www.infoempleo.com/informe-infoempleo-adecco/.

[31] M. de Empleo y Seguridad Social. (ene. de 2018). Bases y tipos de Cotización 2018. Sistema General de la Seguridad Social, [En línea]. Disponible en: http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm#36538 (Acceso: 22-05-2018).

[32] ——, (mayo de 2017). Real Decreto Legislativo 2/2015, de 23 de octubre, por el que se aprueba el texto refundido de la Ley del Estatuto de los Trabajadores., [En línea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-2015-11430%5C&p=20170513%5C&tn=2 (Acceso: 22-05-2018).

[33] G. Kondrak, "N-Gram Similarity and Distance", en *String Processing and Information Retrieval*, M. Consens y G. Navarro, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115-126.

[34] ArcGIS. (), [En línea]. Disponible en: http://desktop.arcgis.com/es/arcmap/10.3/tools/spatial-analyst-toolbox/how-fuzzy-membership-works.htm (Acceso: 21-05-2018).

[35] P. A. V. Hall y G. R. Dowling, "Approximate String Matching", en *Computing Surveys*, ép. nº 4, vol. 12, dic. de 1980, pp. 381-401.

[36] D. Avison y G. Fitzgerald, *Methodologies for Developing Information Systems: A Historical Perspective*. nov. de 2006, vol. 214, pp. 27-38.

ANEXO 1: GLOSARIO

Logica clásica o bivaluada: Aquella que establece el cumplimiento de una condición o pertenencia de un elemento a otro mediante los valores binarios *si* o *no*, sin dar la posibilidad de la existencia de otros valores.

WordNet: Base de datos del idioma inglés que establece relaciones semánticas entre las palabras haciendo uso de la propiedad de sinonimia.

FMQL (Fuzzy Query Mongo Language): Lenguaje usado en el software MongoDB para realizar consultas difusas utilizando etiquetas y comparadores difusos.

JSON: Acrónimo de JavaScript Object Notation. Se trata de un formato de texto ligero para el intercambio de datos.

API: Acrónimo de Application Programming Interface. Se trata de un conjunto de funciones que, a modo de biblioteca, ofrece una manera sencilla de que otro Software la utilice de forma transparente.

API REST: Se trata de un estilo de arquitectura de software que se implementa como una interfaz entre sistemas utilizando directamente HTTP para obtener datos o para indicar las operaciones a ejecutar sobre esos datos.

npm: Ecosistema de librerías que ofrece Node.js para el despliegue de los desarrollos que se realizan con este framework.